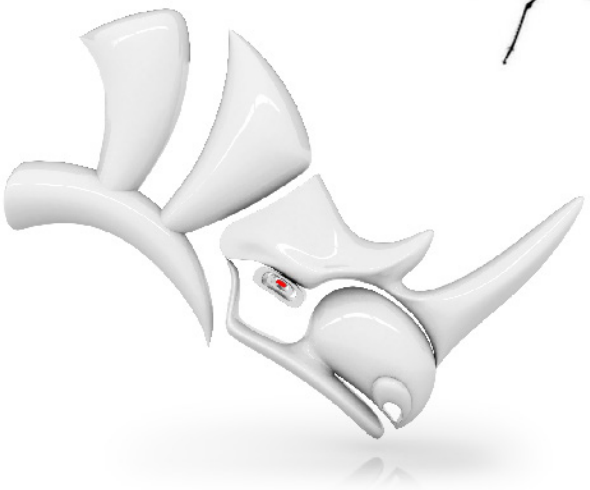
PanelingTools

for **Grasshopper**



PanelingTools helps designers create paneling solutions from concept to fabrication.

Development of the PanelingTools plug-in for Rhino started in 2008. PanelingTools facilitates conceptual and detailed design of paneling patterns using NURBS and mesh geometry. PanelingTools is closely integrated with the Rhinoceros environment using standard Rhino geometry. PanelingTools also extends RhinoScript and Python for completely customized paneling and Grasshopper for parametric modeling.

I am always happy to hear from you and learn how you are using PanelingTools and how to improve it. If you have any questions or suggestions to further its development, feel free to contact me.

Rajaa Issa
Robert McNeel & Associates
Rhinoceros Development team
rajaa@mcneel.com

Chapter 1: Getting Started with PanelingTools

PanelingTools for Grasshopper is under active development. New functionality is added frequently, and like other McNeel products, your feedback is very important and continuously shapes and steers the development.

Note: A working knowledge of the Grasshopper interface is required to be able to use PanelingTools for Grasshopper.

Download and Install

To download PanelingTools for Grasshopper

- Go to *http://www.grasshopper3d.com/group/panelingtools*, and click **Download** to get the latest PanelingTools for Rhino 5 Plugin and PT-GH Add On.

  All PanelingTools instructions, documentation, and discussions are available there.

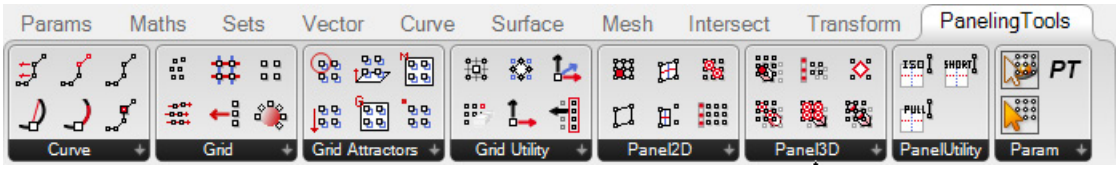
Technical Support

Suggestions, bug reports, and comments are very much encouraged. Please share your stories, examples, and experiences with us. Post questions to our discussion forum *http://www.grasshopper3d.com/group/panelingtools* or e-mail us directly.

Visit *http://www.rhino3D.com/support.htm* for more details.

The Main Menu

When you install PanelingTools, a new **PanelingTools** menu item is added to the Rhino menu bar. You can access all of the PanelingTools commands from there. When you open Grasshopper, a new **PanelingTools** tab is added to the Grasshopper window. It includes all PanelingTools for Grasshopper components.



Toolbars

In addition to the menu, a set of toolbars is installed for PanelingTools for Rhino.

To load the PanelingTools toolbars

1. From the **Tools** menu, click **Toolbar Layout**.
2. Under **Files**, click **PanelingTools**, and in the **Toolbars** list, check the **PanelingTools** box.

Overview of Paneling Elements

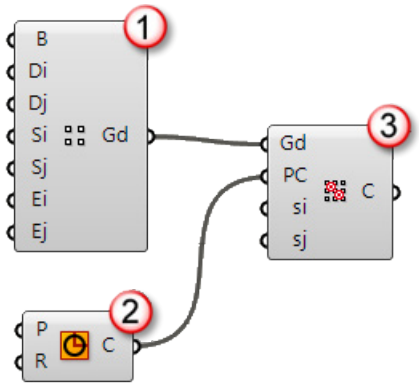
Paneling is typically done in two steps:

1. Create a grid.

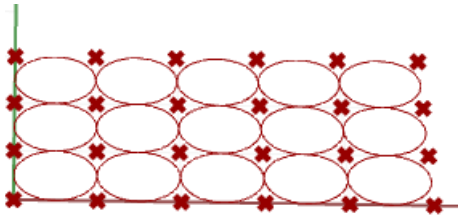
   Create a rectangular paneling grid of points. Creating a paneling grid results in points that can be manipulated with any Grasshopper standard component or PanelingTools Grid Utility components.

2. Populate the grid with paneling elements.

   Populate a pattern or modules of curves, surfaces, and polysurfaces. Generating the paneling creates patterns and applies the patterns to a valid paneling grid of points. The resulting paneling is standard Rhino geometry in the form of curves, surfaces, or meshes. To further process panels with such commands as **Unroll**, **Offset**, **Pipe**, or **Fin**, use Panel Utility components and other Grasshopper components.



Components:
1. *Planar Grid*
2. *Circle*
3. *Morph 2D*



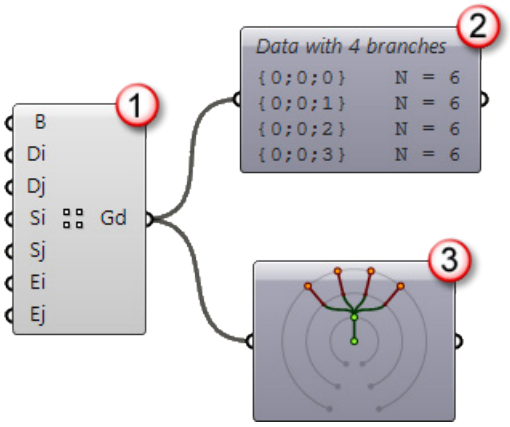
Output Panels.

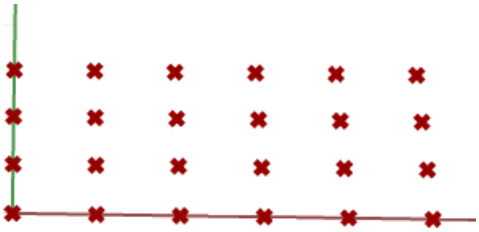
The two-step process gives more flexibility and better control over the result. Normally, the initial grid is generated interactively and is a good indicator of scale. The grid can be generated using the many grid-generating commands or with scripting. The grid can be directly edited and refined before any paneling is applied. Panels can be created using free-form patterns or user-defined patterns that connect grid points.

Create Paneling Grids

A paneling grid is a tree structure of Rhino point objects. For example, a typical output of a single grid is arranged as illustrated below. In the figure, note the following:

- Paneling grids are represented using Grasshopper tree structure.
- Branches represent grid rows.
- Leaves represent points in each row (branches do not have be equal in length).



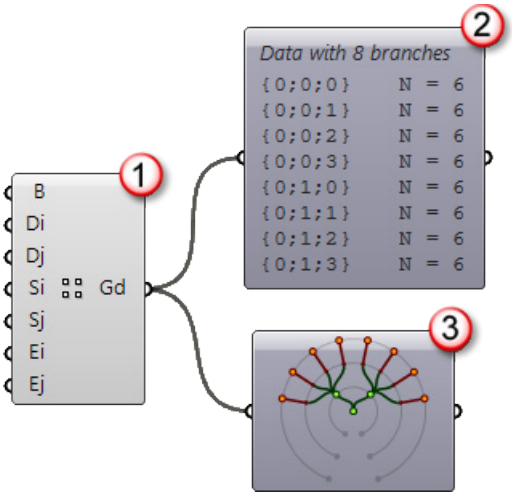


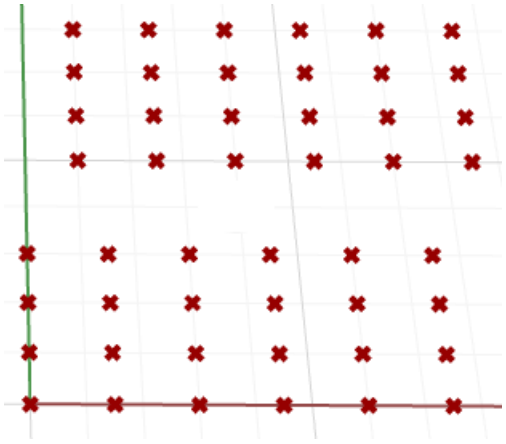
Components:

1. *Planar Grid*
2. *Param Viewer*
3. *Param Viewer Draw Tree*

Output grid.

One tree structure can hold a number of grids as illustrated below. In this case, the input has two base points.





Components:

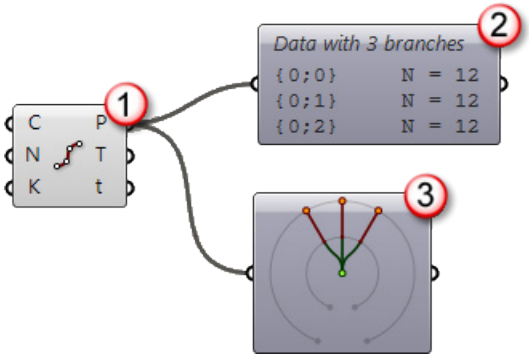
1. *Planar Grid with two base points*
2. *Param Viewer*
3. *Param Viewer Draw Tree*

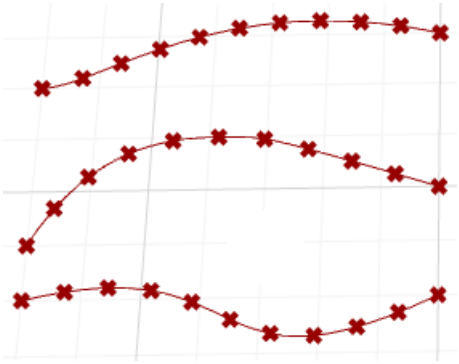
Output grid.

Paneling grids can be generated in many different ways. The following is an overview of these methods.

Create Grids with Grasshopper Standard Components

You can use grids created within a Grasshopper environment as long as the output is a simple tree structure of points where branches represent rows of points. For example, the ***Divide Curve*** component with multiple curve input creates a structure like this making it a valid input for paneling components.



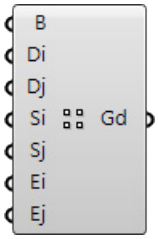


Components:
1. *Divide Curve*
2. *Param Viewer*
3. *Param Viewer Draw Tree*

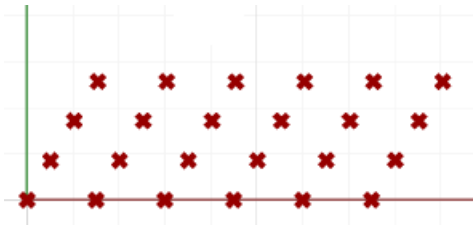
Output grid.

Create Grids with PanelingTools Components

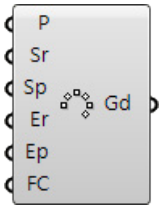
PanelingTools components make a variety of ways to create paneling available whether from scratch or through using reference geometry. The simplest components that do not need a reference geometry are the rectangular or polar grids as illustrated below.



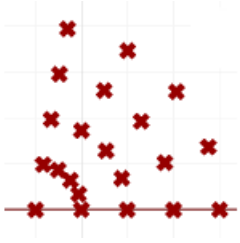
Planar Grid



Output grid.



Polar Grid

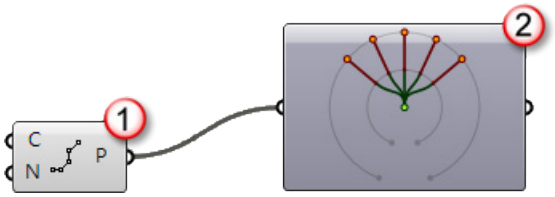


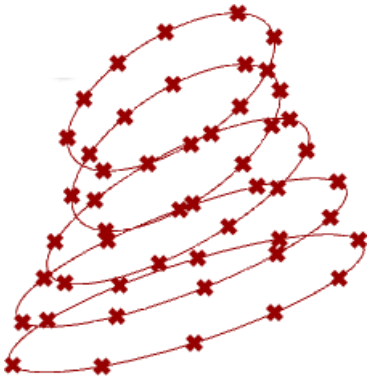
Output grid.

Create a Grid with Reference Geometry

Grids can be based on existing geometry such as curves or surfaces. A variety of grid creation components in PanelingTools can help with that.

For example, you might have an array of curves that we would like our grid to follow.



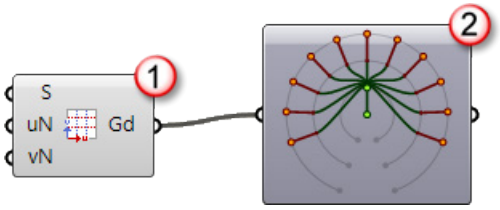


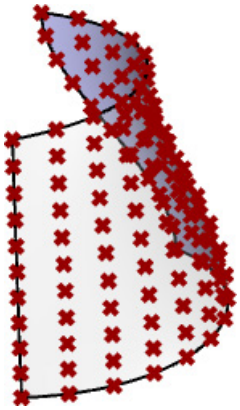
Components:

1. *Divide Number*
2. *Param Viewer Draw Tree*

Output grid.

Or, you might want the grid to follow a surface or a polysurface.





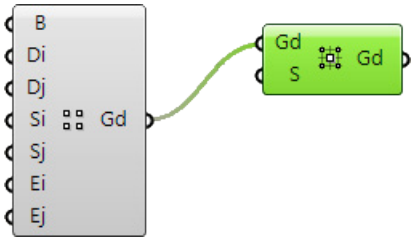
Components:

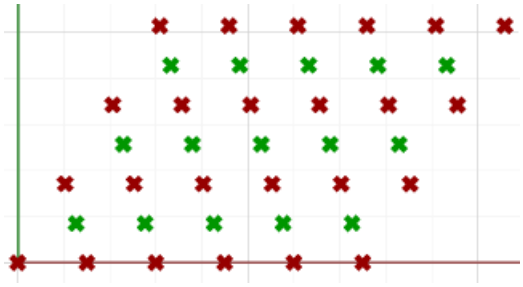
1. *Surface Domain Number*
2. *Param Viewer Draw Tree*

Output grid.

Grid Utilities

PanelingTools provides an array of components that help manipulate the grid as a whole. For example, you might need to flip the direction of the grid (change the base point or swap rows and columns), edit some row directions, or perhaps close the grid in one direction and extend it in another. Many other functions are easier to handle through grid editing components than to do manually. For example, the following is an example of a component that extracts a center grid from an existing grid.

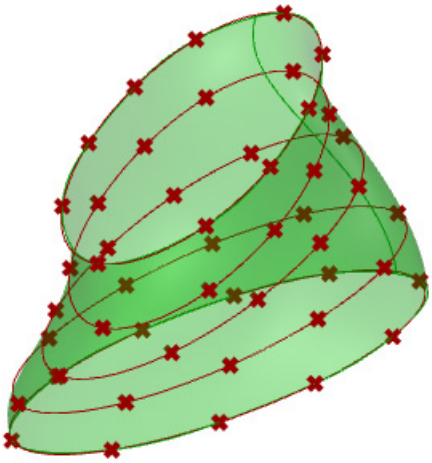




Components:
1. *Planar Grid*
2. *Center Grid*

Output grid.

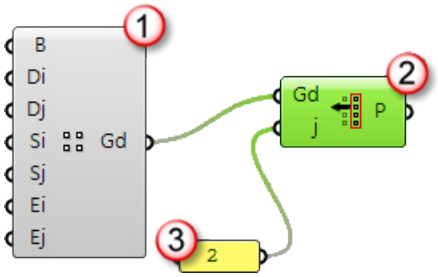
You might want to create a surface that goes through grid points.

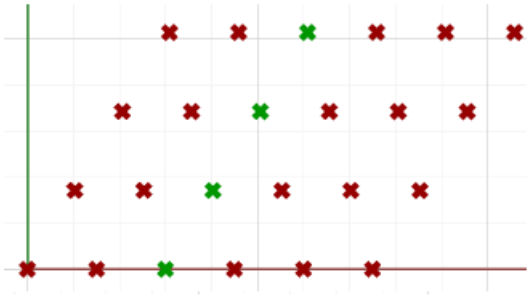


Components:
1. *Divide Number*
2. *Surface from Grid*

Output grid.

Or, extract certain columns or rows from a grid.





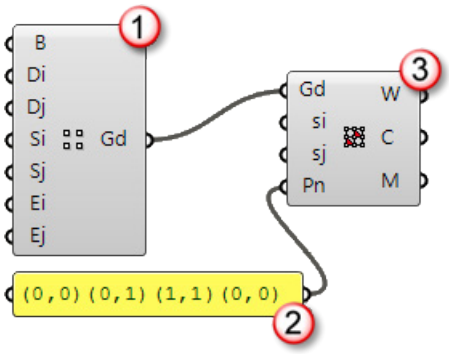
Components:
1. *Planar Grid*
2. *Extract Column*
3. *Panel. With column number to extract*

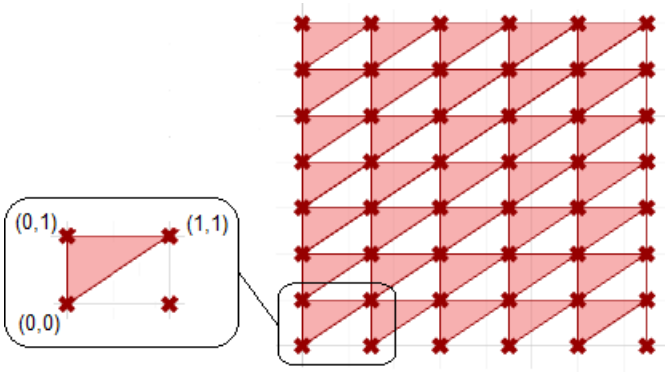
Output column.

Paneling Patterns

In the context of the PanelingTools plugin, *Paneling* refers to the process of mapping geometry or modules to a rectangular grid. Paneling can be either along one grid to generate 2-D patterns or between two bounding grids to generate 3-D patterns. There are three main methods to panel:

1. Connect grid points to create edges, surfaces, or mesh faces of the intended pattern. This approach is the fastest and can cover a wide variety of patterns. You can also use base surfaces to pull the geometry to.





Components:
1. *Planar Grid*
2. *Panel Connections*
3. *Panel. With unit connection string*

Output panels.

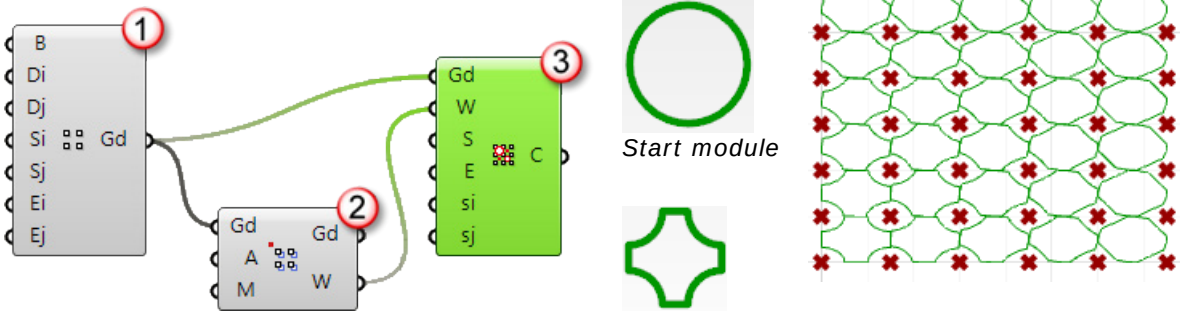
2. Morph a unit module and distribute it over unit paneling grid. This approach can be more time consuming, but allows for rich development of free-form patterns that do not conform to grid points.

Components:
①*Planar Grid*
②*Morph 2D*

2-D curve module.

Output panels.

3. Morph a unit module in a variable way along the grid, depending on design constraints.



Start module

End module

Output panels.

Components:
①*Planar Grid*
②*Point Attraction*
③*Morph 2D Mean*

Attractors as a Design Element

PanelingTools for Grasshopper supports various ways to shuffle grid point locations or distribute variable components based on attractors. Attractors can be points, curves, surface curvature, or other methods. Attractor components calculate the weights of a corresponding input grid and output the attracted point grid as well as the weights grid. Weights range between zero and one, reflecting the degree of attraction for each point in the grid. At a weight of zero, the corresponding grid point is not affected by the attraction. At a weight of one, the corresponding grid point will be affected at the maximum level. The following table displays the available attraction methods:

Attraction Method	Description
Points	Positive values attract towards points, and negative values repel away from points.
Curves	Positive values attract towards curves, and negative values repel away from curves.

Attraction Method	Description
Mean Curvature	Follow the surface mean curvature.
Gauss Curvature	Follow the surface Gaussian curvature.
Vector	Attract relative to an angle with predefined direction vector.
Random	Randomly attract.
Weights	Use an explicit map of attraction values (0-1) per grid point.
Draft Angle	Attract relative to an angle with a plane normal direction.

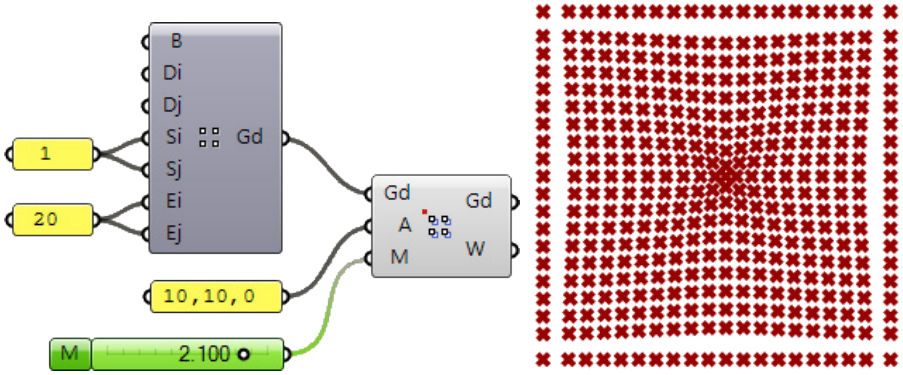
Point Attraction

The input for the ***Point Attraction*** component is a grid of points *(Gd)*, attractor points *(A)*, and a magnitude *(M)*. The output is a shuffled grid of points *(Gd)* and weights *(W)*. If the magnitude value is positive, the points are attracted toward the input points. If the magnitude is negative, the points are repelled. The boundary of the grid is always maintained.

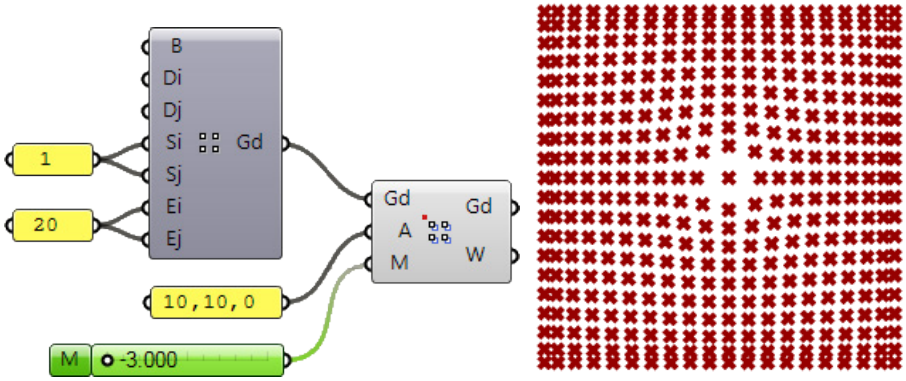
To see an example

- Open ***PT_GH_PrimerExamples\Component Examples\ptPointAtts.gh***.

In the illustration below, the grid points attract towards the center attraction point, since magnitude *(M)* is positive value.

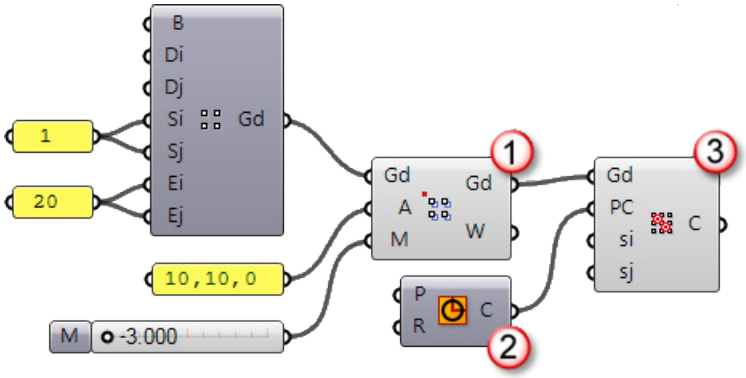


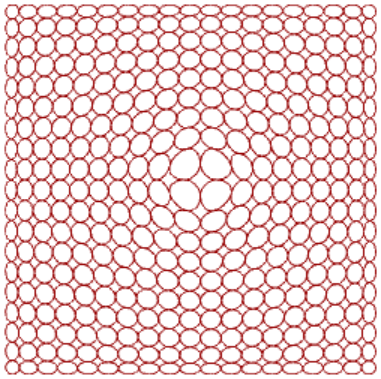
In the illustration below, the grid points are repelled away from the center attraction point, since magnitude *(M)* is negative value.



When you have a shuffled grid, cells will vary in size, and therefore populating a uniform module results in a variable pattern size and shape.

For example, you can then use the attracted grid as input to populate a circle using the ***Morph 2D*** component.





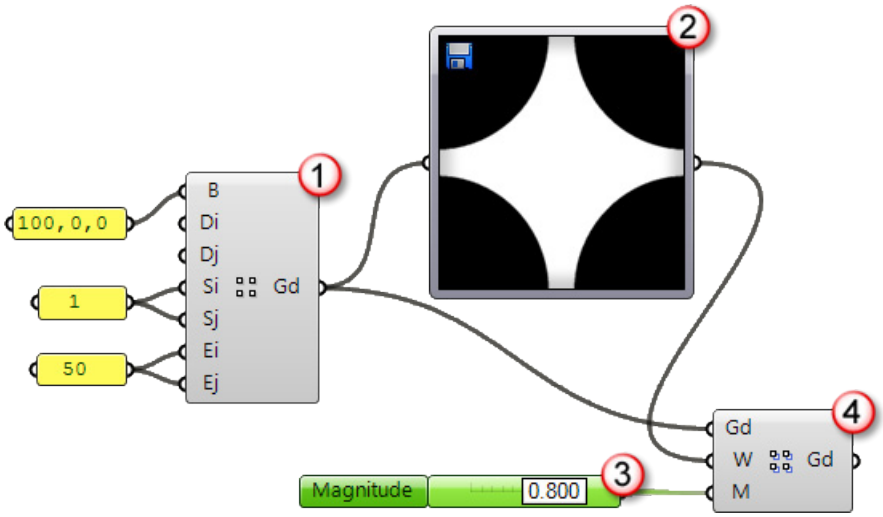
Components:
1. *Point Attraction*
2. *Circle*
3. *Morph 2D*

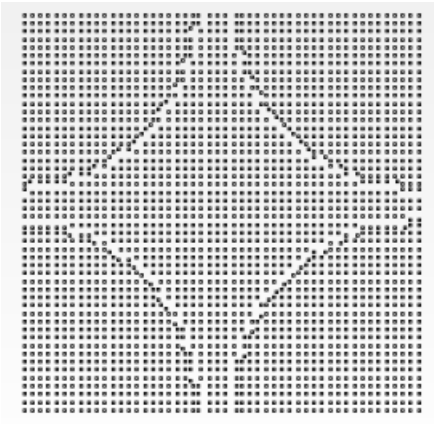
Output panels.

Bitmap Attraction

It is possible to use an image to change locations of grid points. In the following example, the black and white image attracts the paneling grid. Points were moved depending on its greyscale value. The darker the sampled points are, the farther they move.

Using the Grasshopper ***Number Slider*** component, the magnitude value in the ***Weight Attraction*** component can be adjusted to increase or decrease the movement of grid points.





Components:
1. *Planar Grid*
2. *Image Sampler*
3. *Number Slider. Named "Magnitude"*
4. *Weight Attraction*

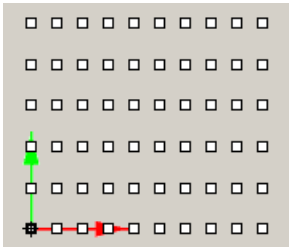
Output grid.

Select a grid

Grids can be generated using the PanelingTools plugin for Rhino (outside Grasshopper). Those grids can be selected as input in Grasshopper.

To select a grid

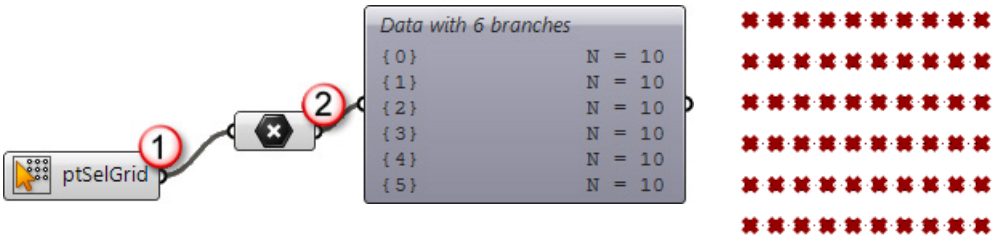
1. Create a paneling grid inside Rhino.
2. On the Rhino **PanelingTools** menu, click **Create Paneling Grid**, and then click **Array**.
3. Change the options to create a grid with **10** points in the x-direction and **6** points in the y-direction.



4. To use this grid as input in our Grasshopper definition, use the ***Select Grid*** component, and then click the icon in the component.

5. Select the grid you just created, and press **Enter**.

   The selected points are organized into six rows with 10 elements in each row.



Components:
1 *Select Grid*
2 *Point*

Output grid.

Bake a Grid

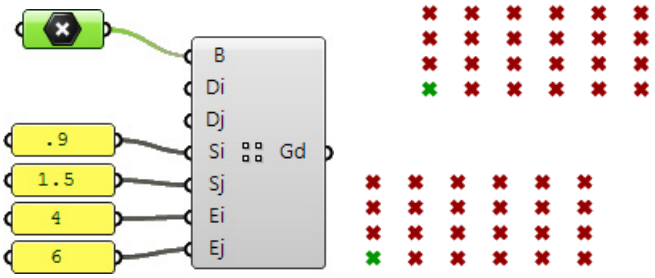
Grasshopper does not create real Rhino objects; geometry is calculated and are then displayed on screen . To add the geometry to the Rhino document, you can *bake* grids created with Grasshopper into Rhino in a format that can be used by PanelingTools commands in Rhino.

Note: Paneling grids in the Rhino document are *ordered* grids. In Grasshopper, you can maintain ordering using the tree structure. In Rhino, PanelingTools tags each grid point with its (row,col) location using the Object Property name. Baking a grid adds points to Rhino and assigns the (row,col) information as an object name to these points.

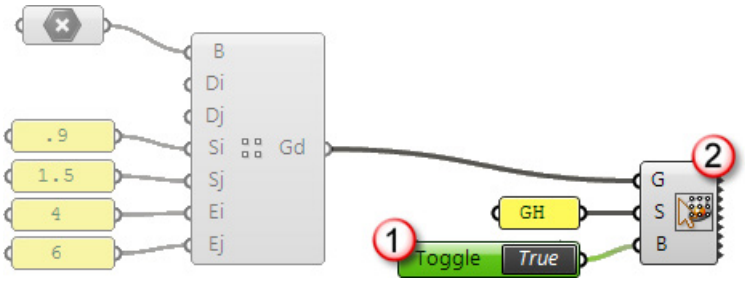
To create actual geometry from the definition

1. Add a ***Planar Grid*** component to the canvas.
2. Pick two base points (B) in the Rhino viewport.
3. Set the shift in the i-direction *(Si)* to **0.9**.

4. Set the shift in the j-direction *(Sj)* to **1.5**.
5. Set the number of points in the i-direction *(Ei)* to **4**.
6. Set the number of points in the j-direction *(Ej)* to **6**.



7. Use the ***Bake Grid*** component to bake the two grids into Rhino by setting the ***Toggle*** to *True*.
8. Be sure set the ***Toggle*** back to *False* so that you do not get multiple bakes whenever the solution is recalculated.



Components:
①*Boolean Toggle*
②*Bake Grid*

Output two grids in Rhino.

Chapter 2: Tutorials

This section introduces a number of tutorials that are meant to show how PanelingTools for Grasshopper is used to create parametric paneling solutions. It should give you a general idea about the context in which these tools may be used.

Diamond Panels

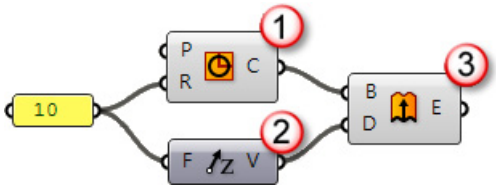
This tutorial introduces two different methods to create diamond panels. The first creates diamond panels by converting a rectangular grid into a diamond grid then paneling the new grid. The second keeps the rectangular grid but defines a connecting pattern for the diamond panels. Both are valid approaches, and you can choose the one that works best with your workflow.

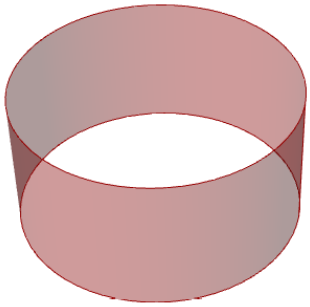
To see an example

- Open ***..\PT_GH_PrimerExamples\Tutorial Examples\DiamondPanels.gh***.

Create diamond panels

1. Create an uncapped cylinder using Grasshopper standard components.



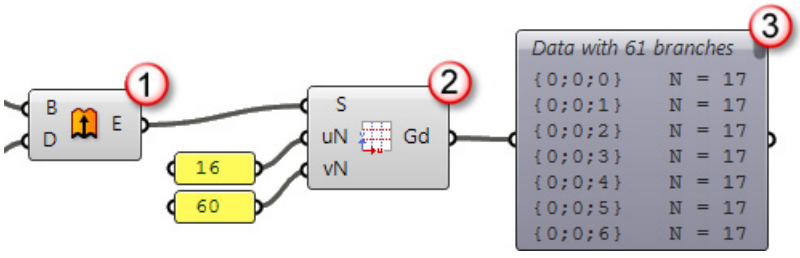


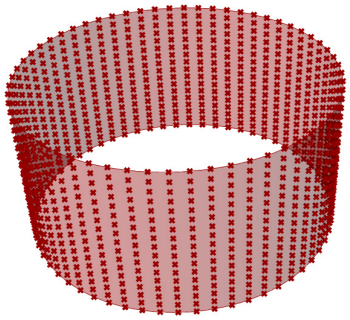
Components:
1. *Circle*
2. *Unit Z*
3. *Extrude*

Output cylinder.

2. Create a rectangular grid of points using the ***Grid > Surface Domain Number*** component.

   The grid distribution follows the isocurve direction of the underlying surface. In this example, the u-direction of the surface is vertical and hence the row directions are vertical. Each row has 16 spans (17 points). Columns are in the circular direction and each column has 60 spans (61 points). The first and last points in each of the 17 columns overlap because the surface is a closed surface.

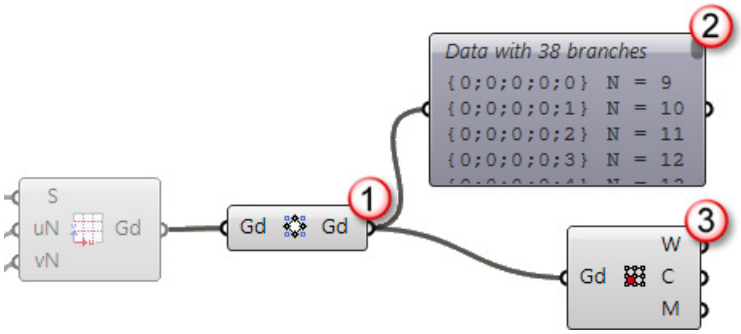


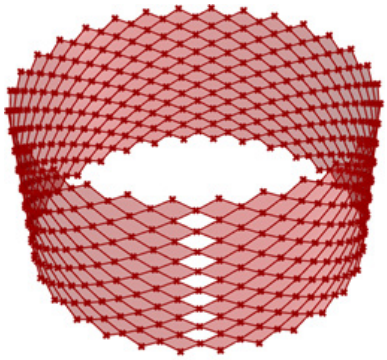


Components:
1. *Extrude*
2. *Surface Domain Number*
3. *Param Viewer*

Output grid.

3. Following the first method of creating diamond panels, you can use the ***Grid Utility > Convert to Diamond*** component to extract a new rectangular grid in the diagonal direction and then use the ***Panel2D > Cellulate*** component to create the panels.



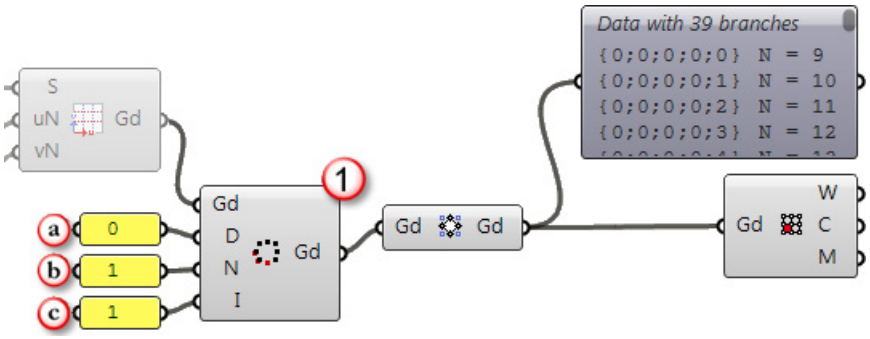


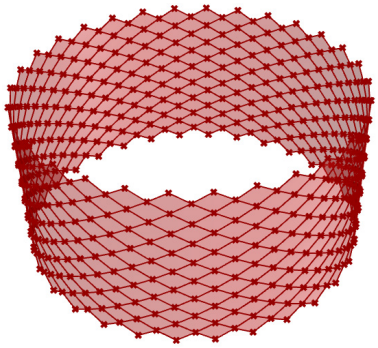
Components:

1. *Surface Domain Number*
2. *Param Viewer - rows of the diamond grid have a variable number of elements*
3. *Cellulate - creates panels*

Output panels.

Notice that there are missing panels along the seam. This is because the pattern effectively ran out of grid points to cover. To deal with this situation, you need to wrap the grid so that one extra row overlaps the second row. The first and last rows already overlap, so we just need one extra row. Use the ***Grid Utility > Wrap Grids*** component.





Components:

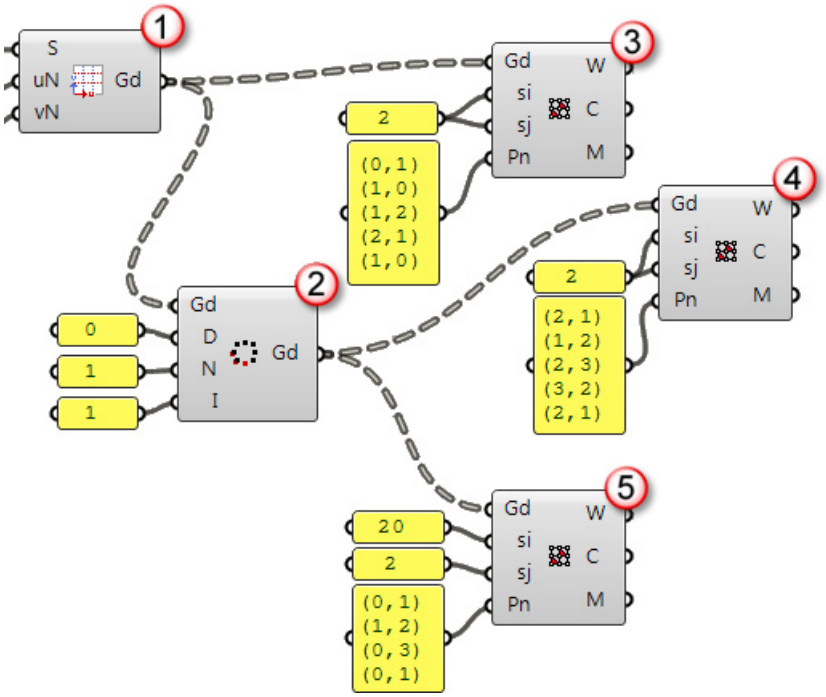
1. *Wrap Grids*

Parameters:

a. *Wrap direction (D)*
b. *Number of rows/columns to wrap (N)*
c. *Starting index (I)*

Output panels.

Another approach to creating diamond panels is to use the ***Panel2D > Panel Connections*** component. We still need to create the grid and wrap it one extra row. The following example illustrates how the definition works. Each of the three ***Panel Connections*** components accepts a grid *(Gd)*, a shift in the u-direction *(si)* and in the v-direction *(sj)*, and a string *(Pn)* that represents the uv-unit points each pattern connects.

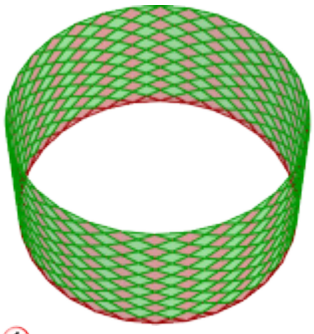


Components:

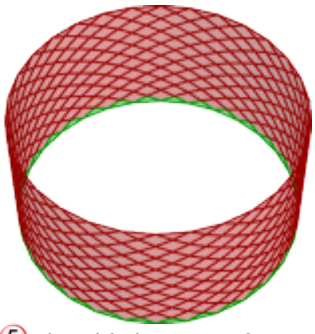
1 *Surface Domain Number*

2 *Wrap Grids*

3 *The first group of diamond panels*



4 *The second group of diamond panels*



5 *The third group of panels along the edge*

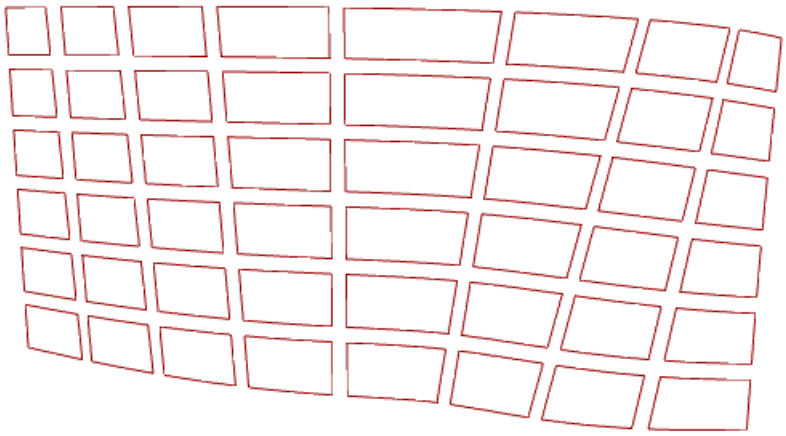
Fixed Gaps Between Panels

This tutorial shows how to achieve a fixed gap between panels that are based on a free-form surface.

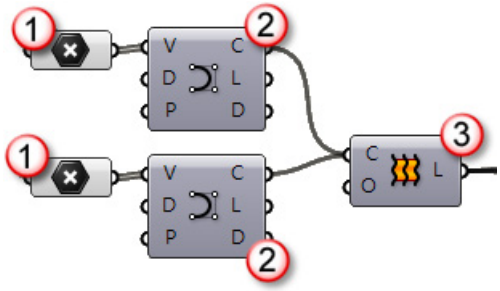
To see an example

- Open ***..\PT_GH_PrimerExamples\Tutorial Examples\FixGap.gh***.

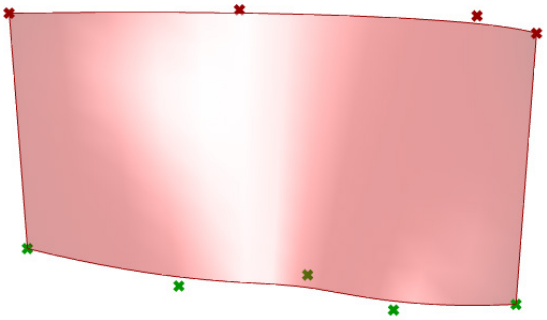
Note: In the example file, the **Preview** is turned off for all but the final paneling result. Components colored dark gray have their **Preview** turned off. To see the output on the intermediate steps, turn the **Preview** on for the components.



1. Start the definition by creating a free-form loft surface from two NURBS curves (***Grasshopper > Surface > Freeform > Loft***).

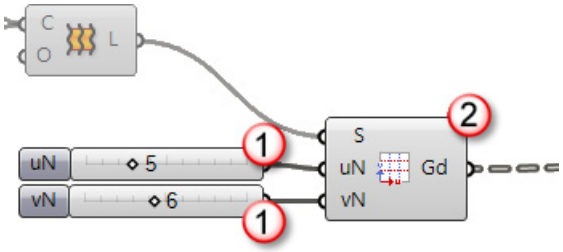


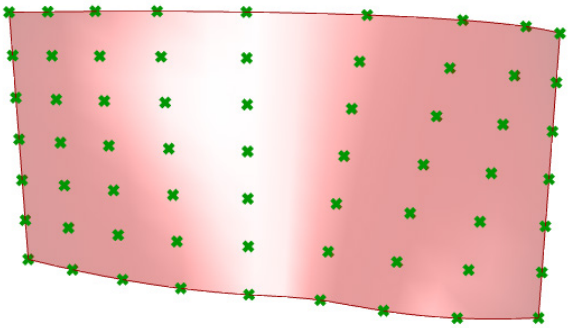
Components:
1. *Vertices - control points for loft curves*
2. *Curve. Generated loft curves*
3. *Loft*



Generated loft surface.

2. Create a grid on the surface using the ***PanelingTools > Grid > Surface Domain Number*** component.

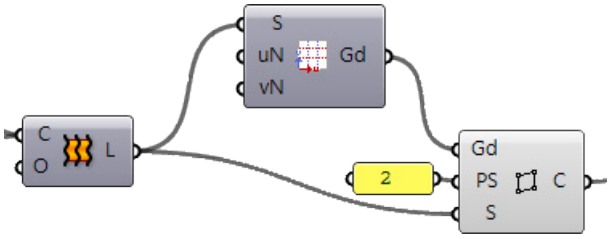


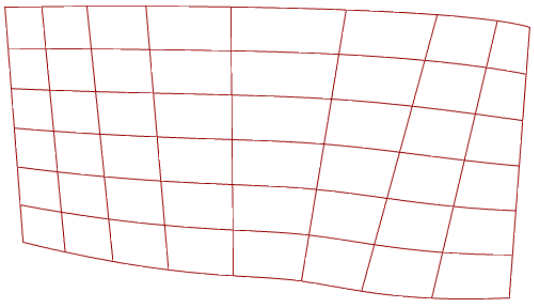


Components:
①*Number Slider. Number of points in u- and v-directions*
②*Grid. Generated grid*

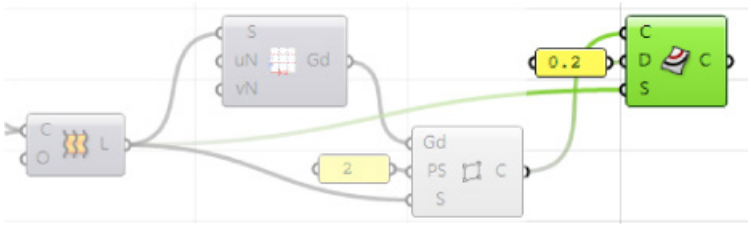
Generated grid on surface.

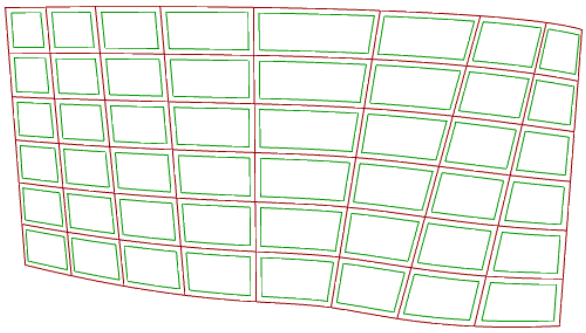
3. Use the ***PanelingTools > Panel2D > Generate Borders*** component to get the panels as a polycurve outline.





4. Use the ***Grasshopper > Util > Offset on Srf*** component to offset the panel outline by a fixed distance on the surface.

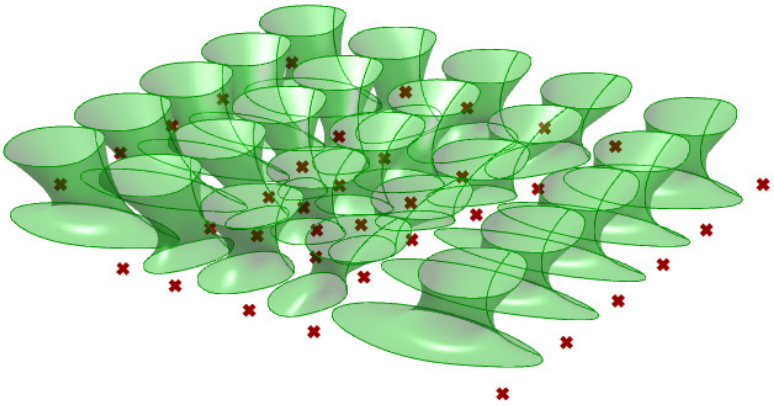




Loft Morphed Curves

This tutorial shows how to:

- Create an attracted grid
- Morph module curves in 3-D space
- Create 3-D modules from morphed curves

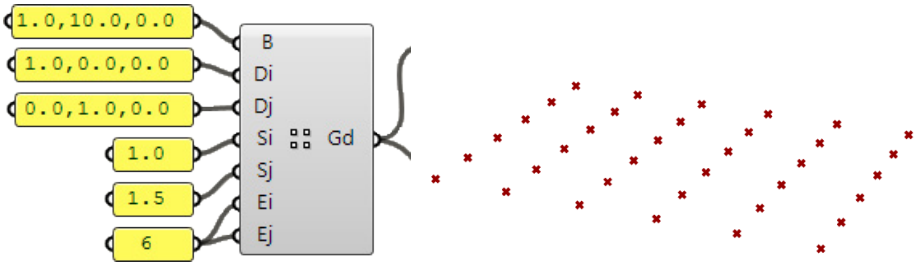


To see an example

- Open **..\PT_GH_PrimerExamples\Tutorial Examples\LoftMorphedCurves.gh**.

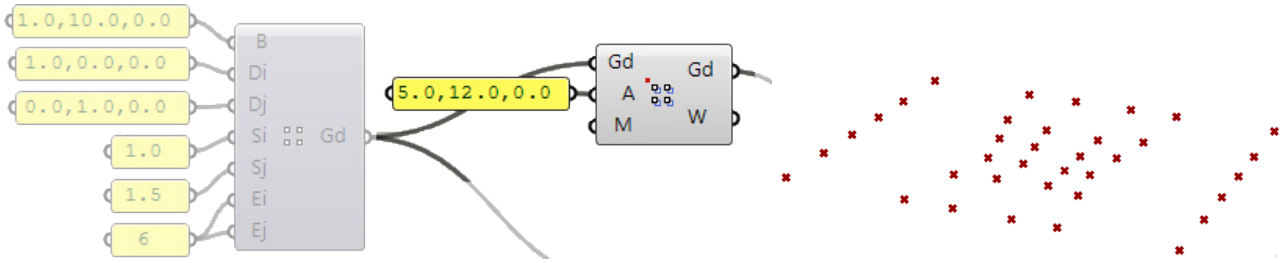
To create the grid

1. Start the Grasshopper definition by creating a rectangular grid using the ***PanelingTools > Grid > Planar Grid*** component.



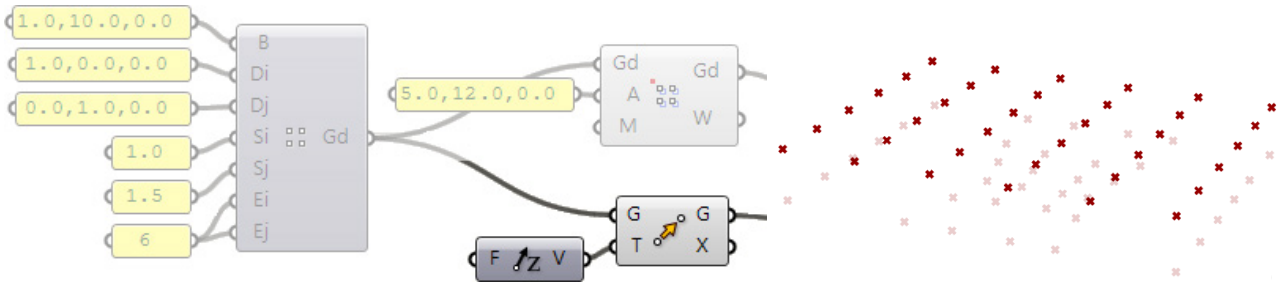
Parameters:
B (base point) = 1.0,10.0,0
Di (row dir) = 1.0,0.0,0.0
Dj (col dir) = 0.0,1.0,0.0
Si (row spacing) = 1.0
Sj (col spacing) = 1.5
Ei (row number) = 6
Ej (col number) = 6.

2. With the ***PanelingTools > Grid Attractors > Point Attraction*** component, add an attractor point and change the grid points to attract towards the attractor point.



Parameters:
Gd (grid) = grid to be attracted
A (attractor point) = 5.0,12.0,0.0
M (Magnitude of attraction) = 1
Gd (output) = attracted grid
W = weights grid (attraction degree for each grid point 0-1)

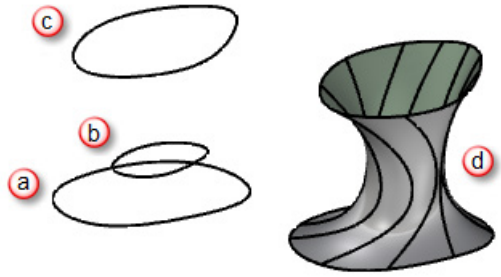
3. Copy the original planar grid in the z-direction to create a second bounding grid to populate the module between the two grids.



Parameters:
G = input geometry
T = input vector

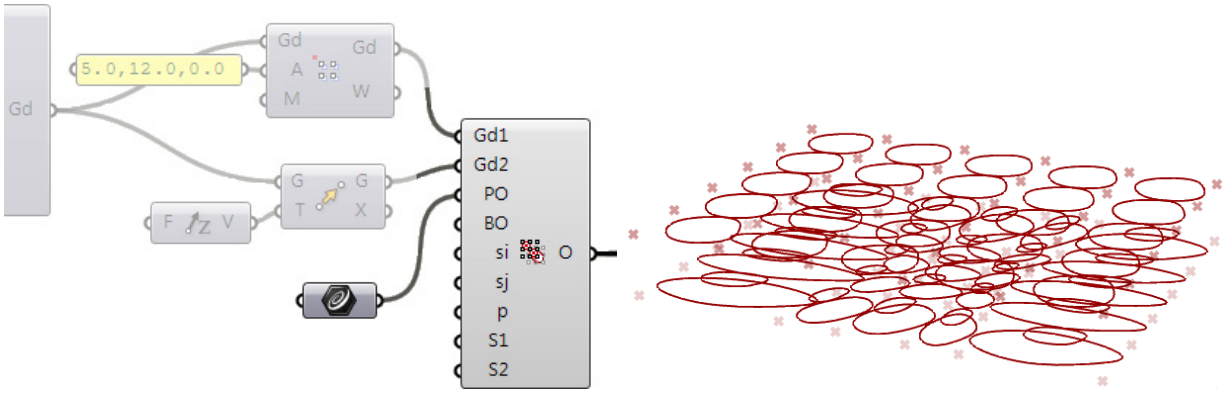
4. Create the module curves in Rhino.

In this case, three curves define a lofted surface. We will morph the curves rather than the lofted surface because it is faster and more efficient.



ⓐ *First module curve*
ⓑ *Second module curve*
ⓒ *Third module curve*
ⓓ *Lofted surface*

5. Reference the module curves in the next step to morph between our two bounding grids using the ***PanelingTools > Panel3D > Morph 3D*** component.



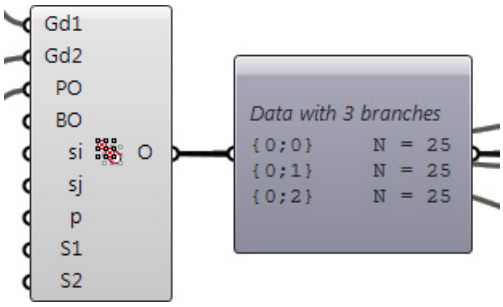
Parameters:
Gd1 = first bounding grid
Gd2 = second bounding grid
PO = pattern objects
BO (optional) = bounding objects for the pattern objects
si = shift in the i direction = 1 (default)
sj = shift in the j direction = 1 (default)
p = pull for smooth morphing = false (default)
S1(optional) = grid1 surface
S2 (optional) = grid2 surface

Note: The output curves from the ***Morph 3D*** component are organized into three branches.

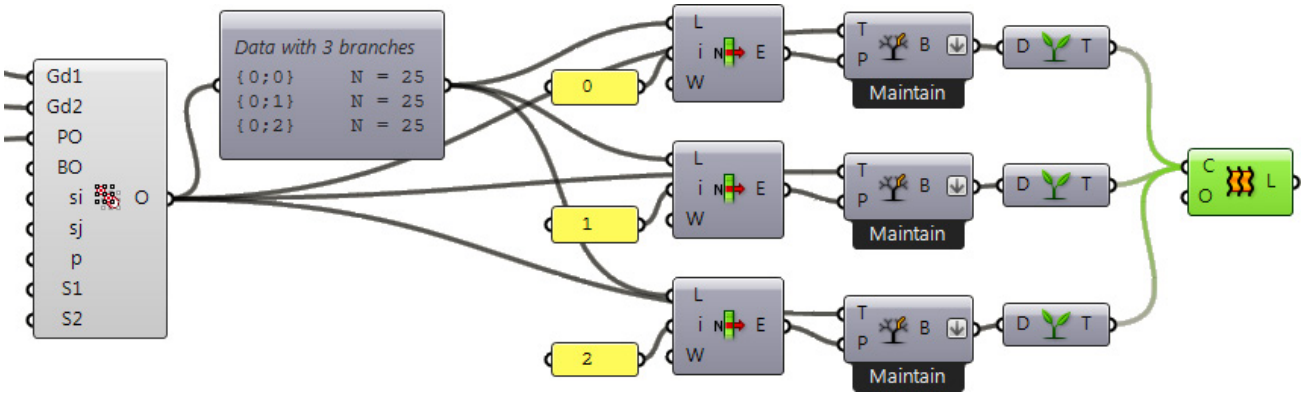
{0;0} holds morphed a curves.

{0;1} holds morphed b curves.

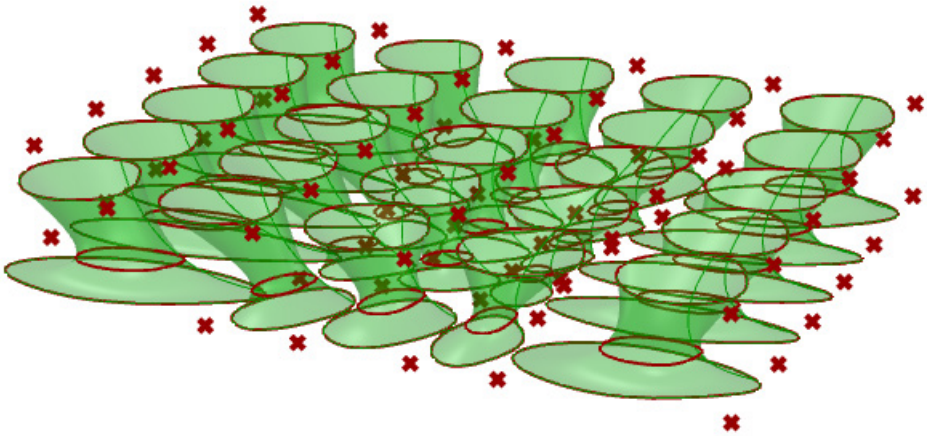
{0;2} holds morphed c curves.



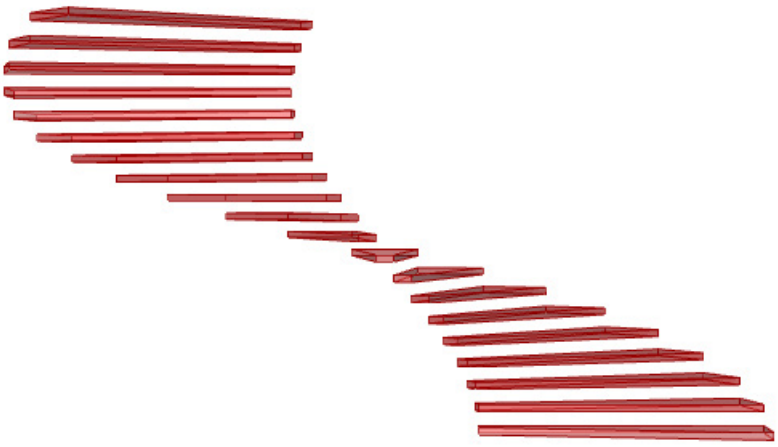
6. Separate the three branches of the tree, then graft each branch before feeding into the ***Grasshopper > Freeform > Loft*** component as illustrated.



This how the lofted modules look:



Spiral Staircase

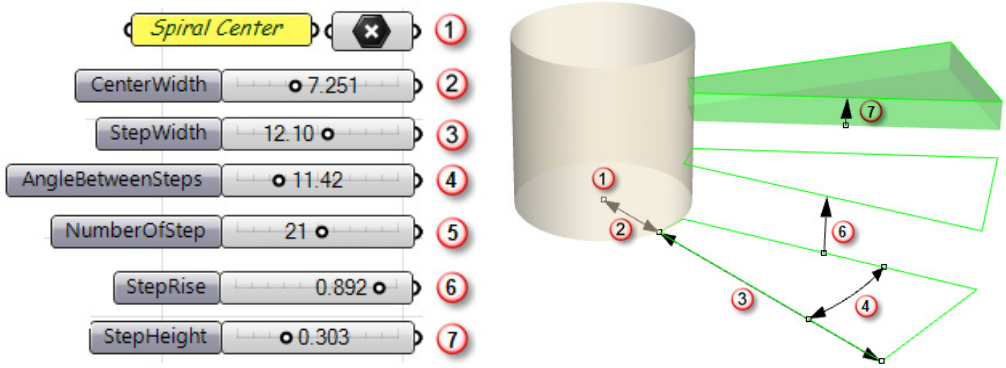


This tutorial shows how to create parametric steps for a spiral staircase.

To see an example

- Open ***..\PT_GH_PrimerExamples\Tutorial Examples\SpiralStaircase.gh***.

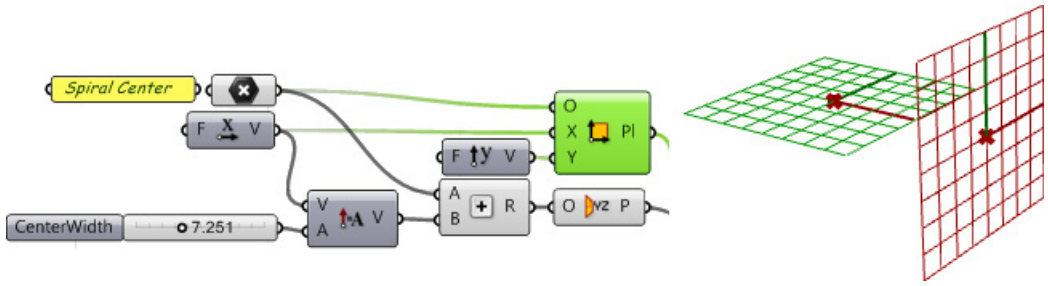
The parameters that control the shape of the spiral staircase are as follows:



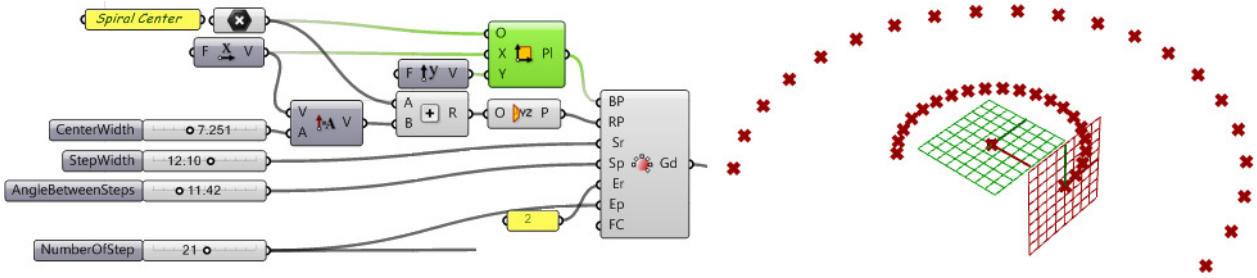
1. *Center of the spiral staircase*
2. *Radius of the center hole*
3. *Step width*
4. *Angle between steps*
5. *Number of steps*
6. *Step rise*
7. *Step height*

Create a spiral staircase

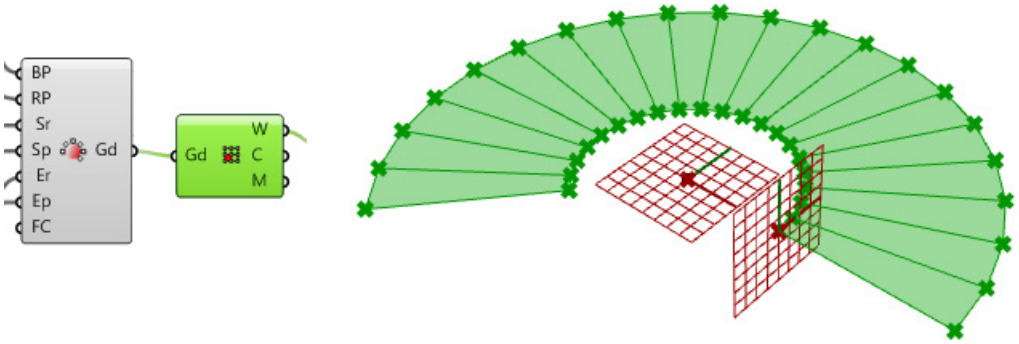
1. Define the position of the staircase and the width of the central cylindrical hole using two planes.



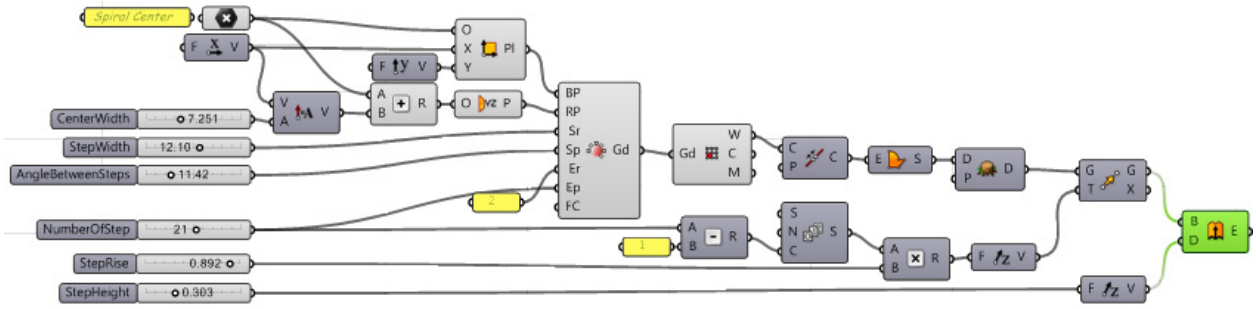
2. Generate the base grid for the stairs using the ***PanelingTools > Grid > Polar 3D Grid*** component.

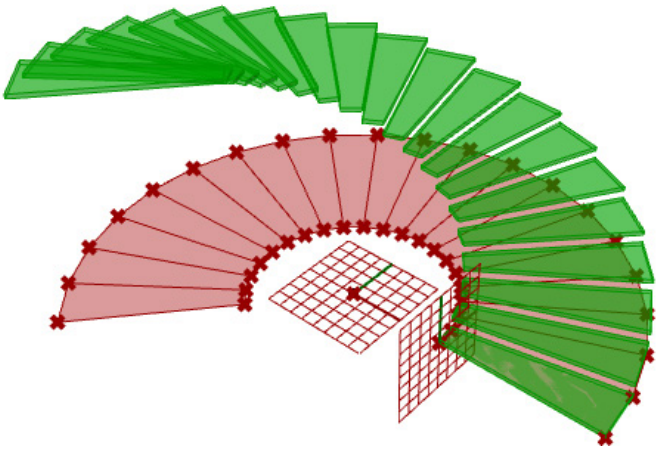


3. Create the base panel for the stairs using the ***PanelingTools > Panel2D > Cellulate*** component.



The remaining steps involve using Grasshopper components to raise the stairs and add thickness to them.

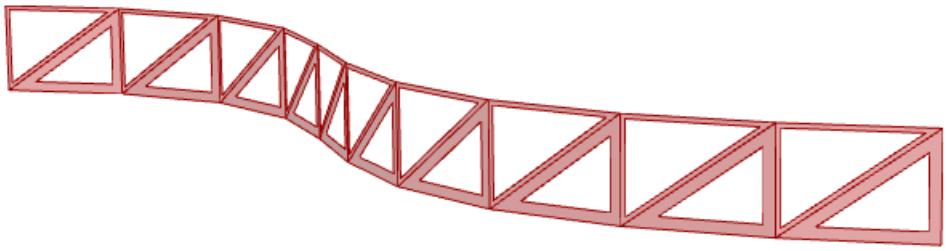




Parametric 2-D Truss

This tutorial shows how to create a parametric truss based on a curve. The truss is based on David Fano's truss tutorial. The main advantages of using PanelingTools-Grasshopper over Grasshopper standard components are:

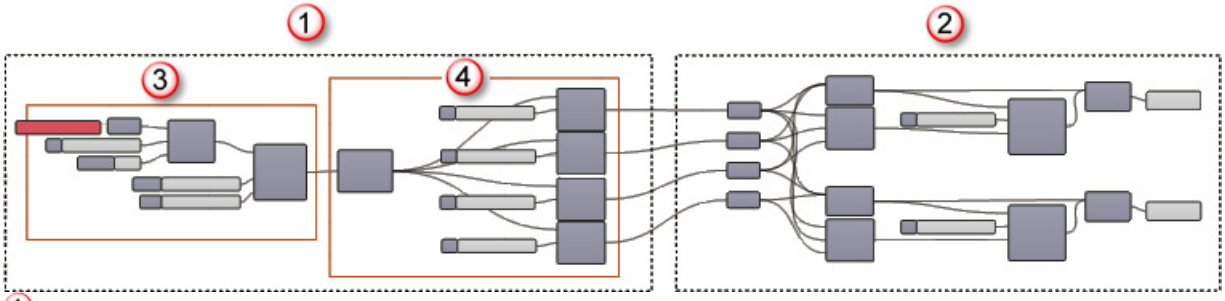
- The system logic is easier to understand, create, and edit.
- The system logic is more flexible. It is not restricted to surfaces and their isocurve directions, which greatly limit control over dimensions and orientation of truss components.
- The truss component logic is based on points, rather than surfaces, which is lighter.



The overall definition is structured into two parts: the system logic and the component logic. The component logic uses standard Grasshopper components based on four corner points. The system logic defines a rectangular grid of cells using PanelingTools-Grasshopper components.

To see an example

- Open ***..\PT_GH_PrimerExamples\Tutorial Examples\Truss.gh***.

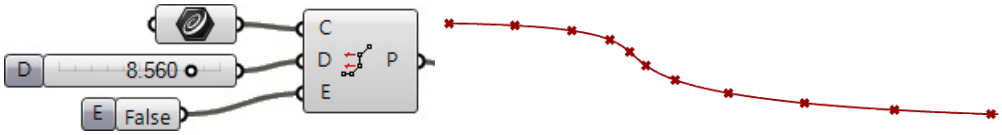


1. *System logic*
2. *Component logic*
3. *Create system grid*
4. *Extract components corners in the system*

Create a parametric truss

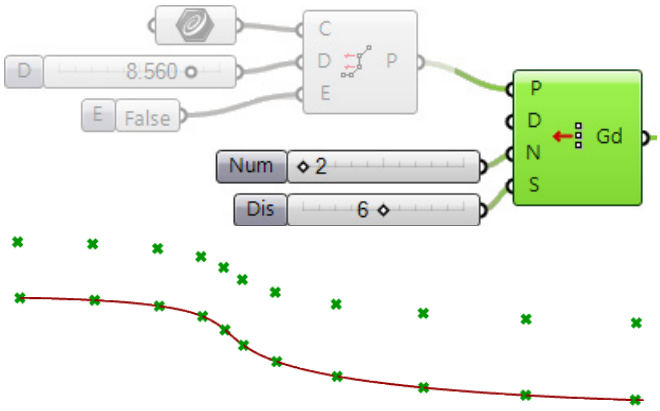
1. Create a reference a curve in Rhino.
2. Divide the curve by a distance that represents the width of the truss.

   Note: PanelingTools for Grasshopper provides a variety of ways to generate a basic grid of cells using the **Grid** panel or simply by feeding a tree structure of points using Grasshopper standard components.



3. Place a ***PanelingTools > Grid > Planar Extrude*** component on the canvas.

   Grid components generate two-dimensional grids of points and organize them into a simple Grasshopper tree structure where each branch contains a list of points representing grid rows.



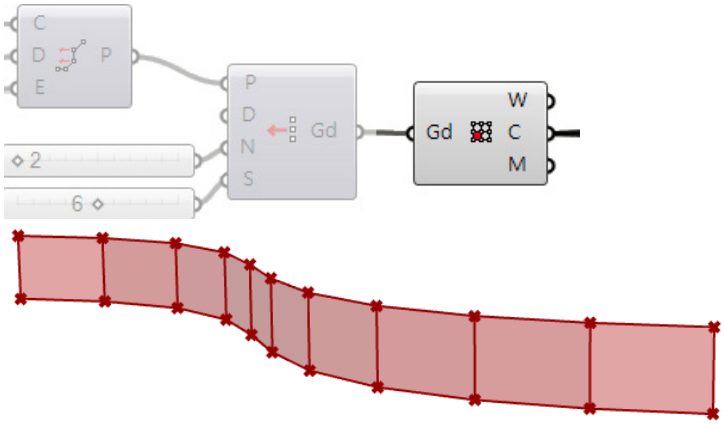
4. Use the ***PanelingTools > Panel2D > Cellulate*** component to extract individual cells of the grid.

It outputs three components:

W (wires): a list of all edges

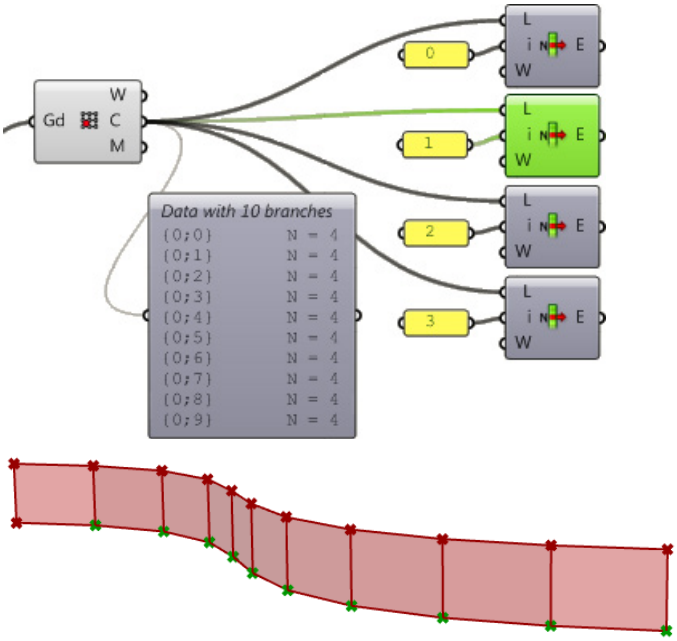
C (cells): a list of the four corners of each cell (this is what we need here)

M (meshes): a list of mesh faces of all cells



We now have 10 cells; each has four corners. We need to get a separate list of each corner to feed into our component logic.

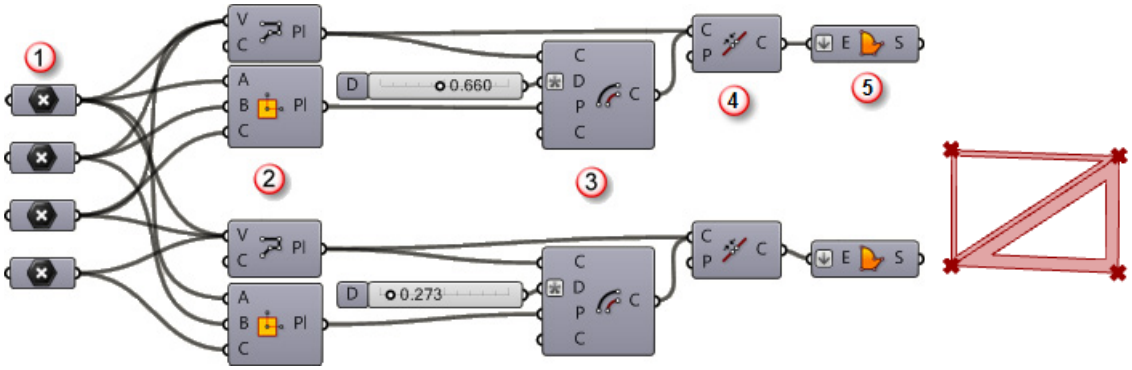
5. Use the Grasshopper **List** component to separate into four lists of corners.



6. Create the component logic of the truss units based on four points.

7. Divide this into two triangles.

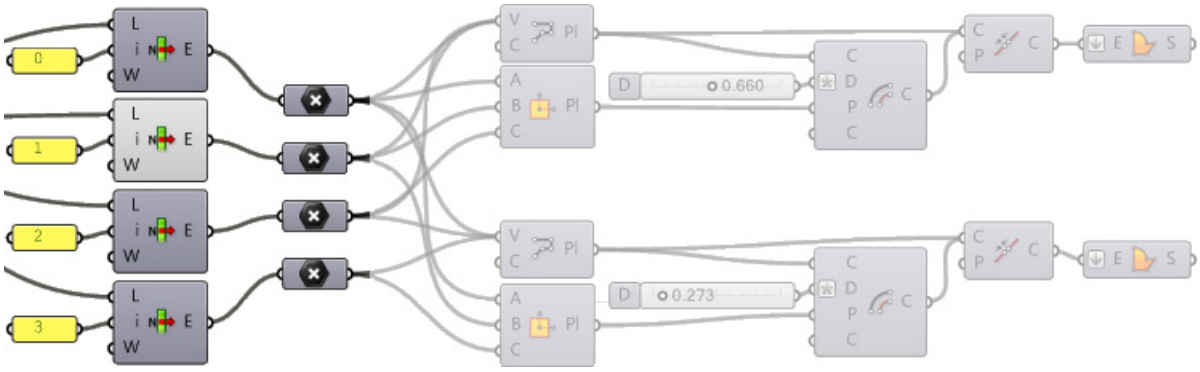
Each triangle can have its own thickness and creates a trimmed planar surface.



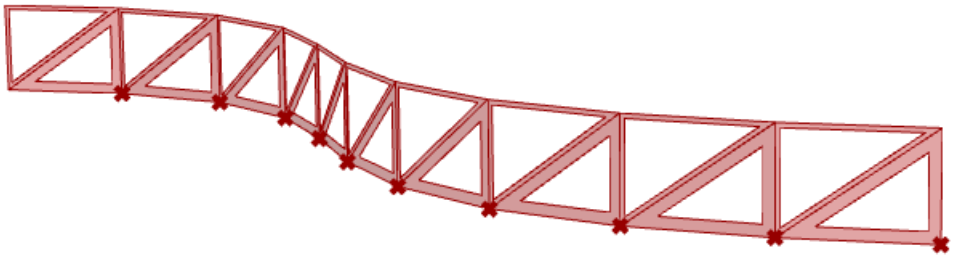
Components:

1. *Truss unit corners*
2. *Upper and lower triangle polylines*
3. *Offset triangles by distance specified in the slider*
4. *Join*
5. *Create planar surface*

8. Hook the system points into the custom truss component logic that is based on four points.



This is what the final truss looks like.

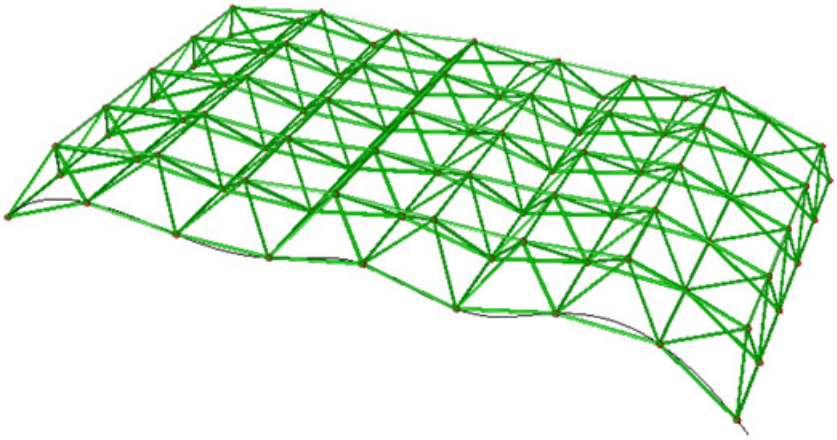


Parametric Space Frame

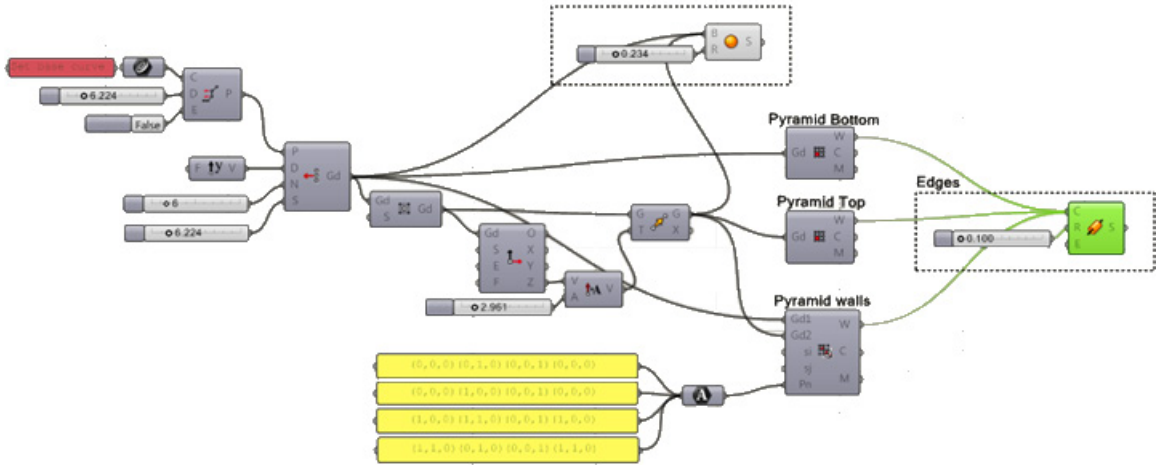
This tutorial shows how to create a parametric space frame based on a curve. While it is possible to create a similar definition using standard Grasshopper components, using PanelingTools for Grasshopper makes the definition easier to write, read, and edit. It also enables better control over dimensions and shape.

To see an example

- Open ***..\PT_GH_PrimerExamples\Tutorial Examples\SpaceFrame.gh***.

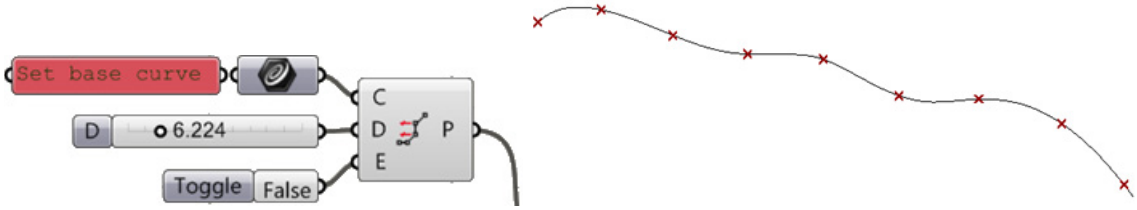


This is how the final definition looks:

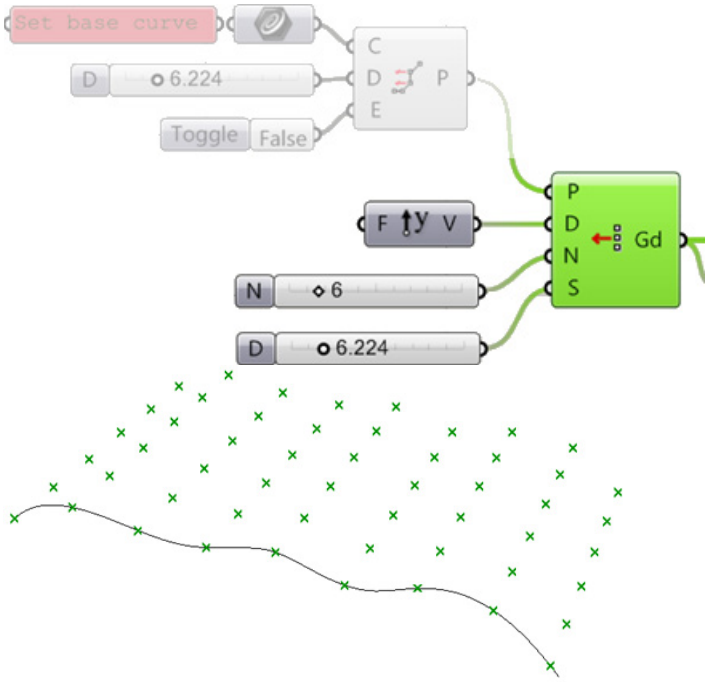


Create a space frame

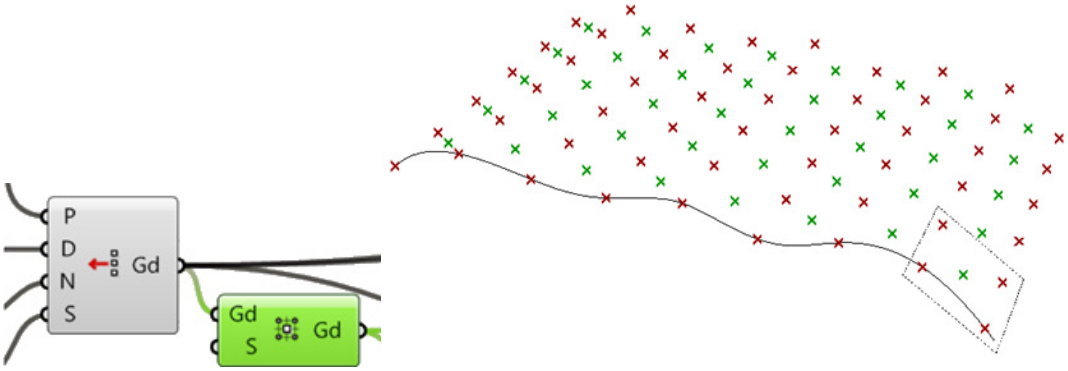
1. Use the ***PanelingTools > Curve > Divide Distance*** to divide a curve by a distance that represents the width of the base cells of the space frame.



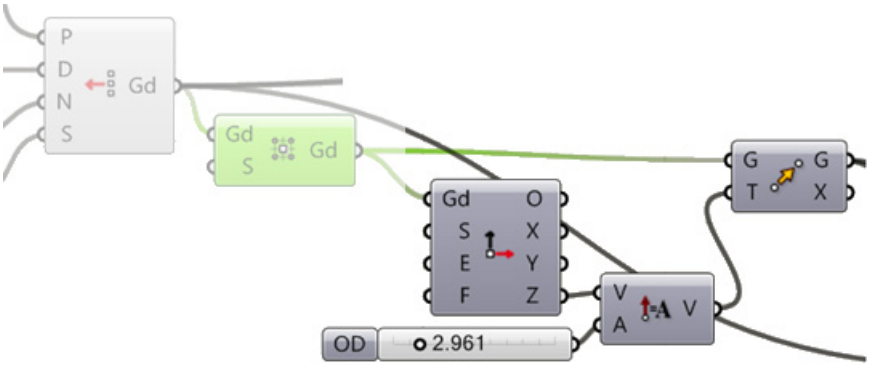
2. Generate the grid using the ***PanelingTools > Grid > Planar Extrude*** component.



3. Extract the center grid using the ***PanelingTools > Utilities > Center Grid*** component.

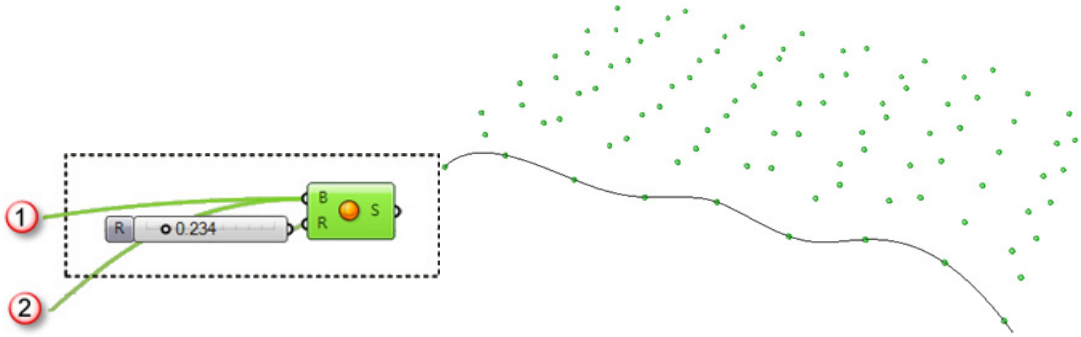


4. Move the center grid in the normal direction to create the top bounding grid of the space frame. In order to move the grid in the normal direction, use the coordinate component that extracts origin of each grid cell and x, y and z directions.



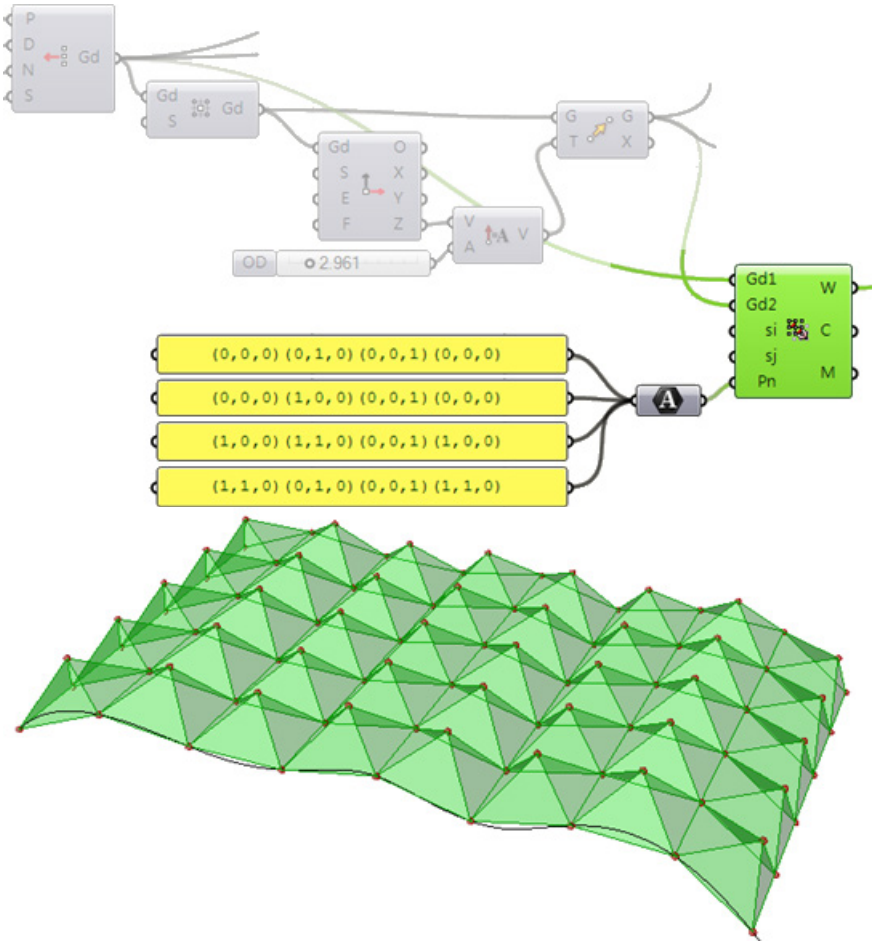
5. Use Grasshopper ***Sphere*** component to create small spheres to mark the joints.

Input both the base grid and moved center grid.

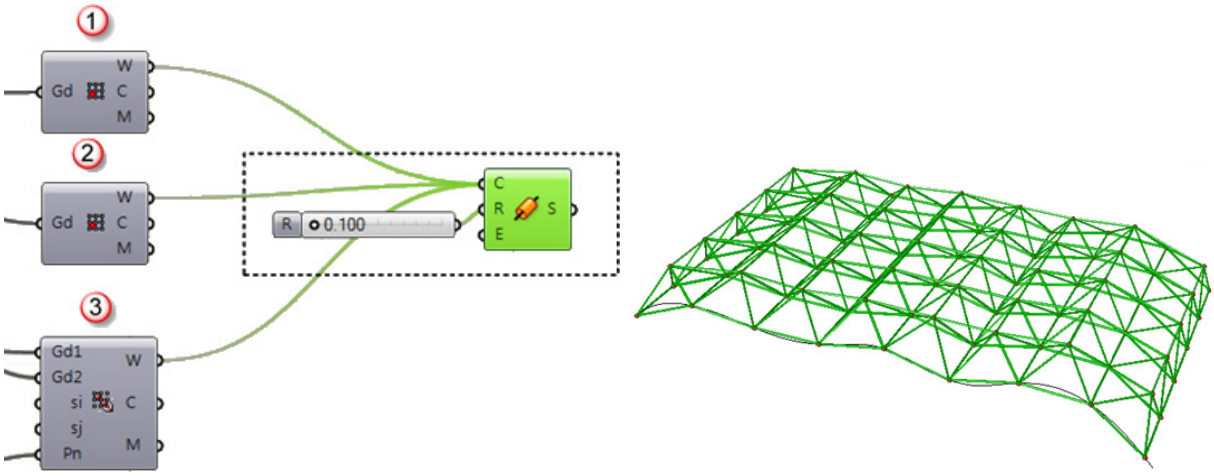


6. Use the ***PanelingTools > Panel3D > Cellulate 3D Grid*** component to generate the bottom and top wires.

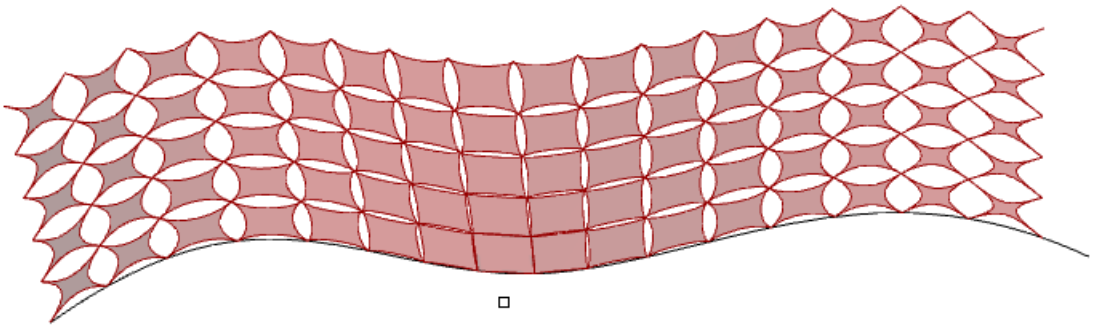
Wall wires and faces of the space frame are created with the Panel 3D Connections component where each face is defined by a string. The string defines indices of the grid to connect. These connections are repeated throughout the grid.



7. Wires from the top, bottom, and walls are then used to generate edge pieces of the space frame using the Grasshopper ***Cylinder*** component.



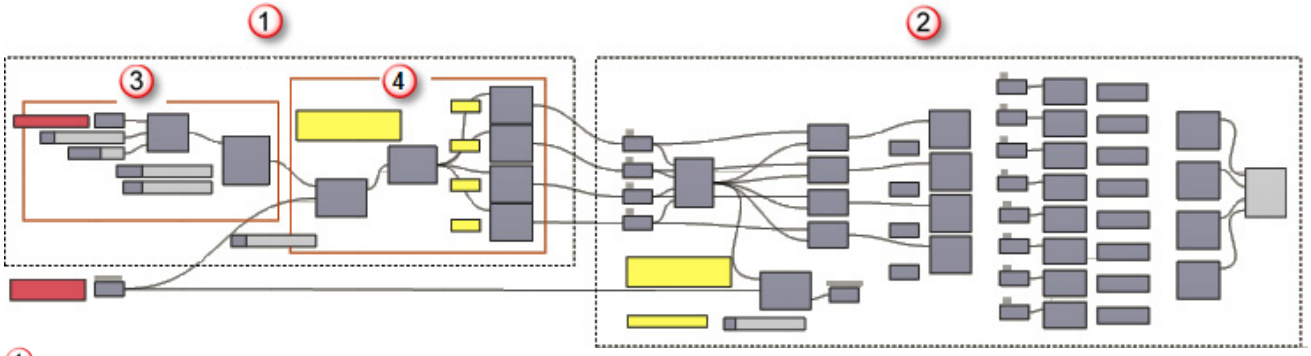
Variable Wall



This tutorial shows how to create a parametric wall with variable system and variable components based on a curve. The ***Wall*** component logic is based on four-corner points system. It uses a PanelingTools attractor component to create a variable component. The system logic defines a rectangular grid of cells. The system has variable cell size using the ***PanelingTools > Grid Attractors > Curve Attraction*** component.

To see an example

- Open ***..\PT_GH_PrimerExamples\Tutorial Examples\Wall.gh***.

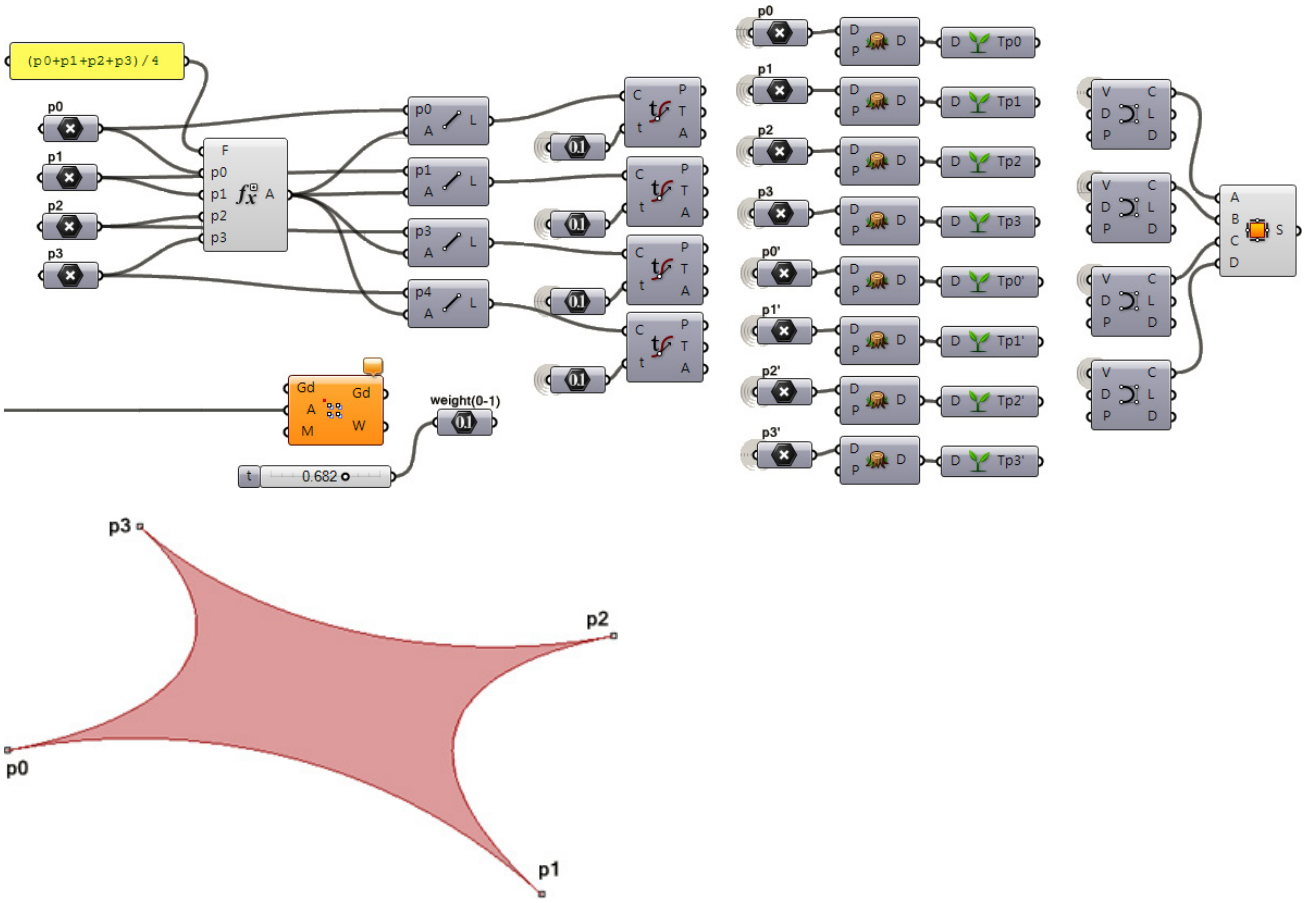


1. *System logic*
2. *Component logic*
3. *Create system grid*
4. *Extract components corners in the system*

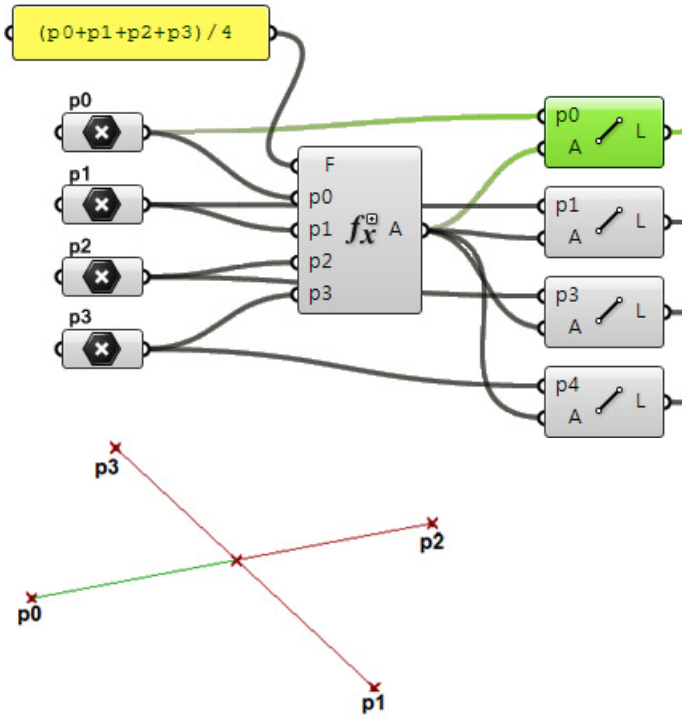
Create a parametric wall

1. Start with building the component logic.

   The following image shows an overview of the component logic based on four corner points and weights that controls the shapes of edge curves. Keep in mind that once the component logic is hooked to the system logic, there will be a list of corner points instead of just one. This is why we need extra steps like flattening and grafting the lists to get correct results. This is how the final component logic defition looks like.

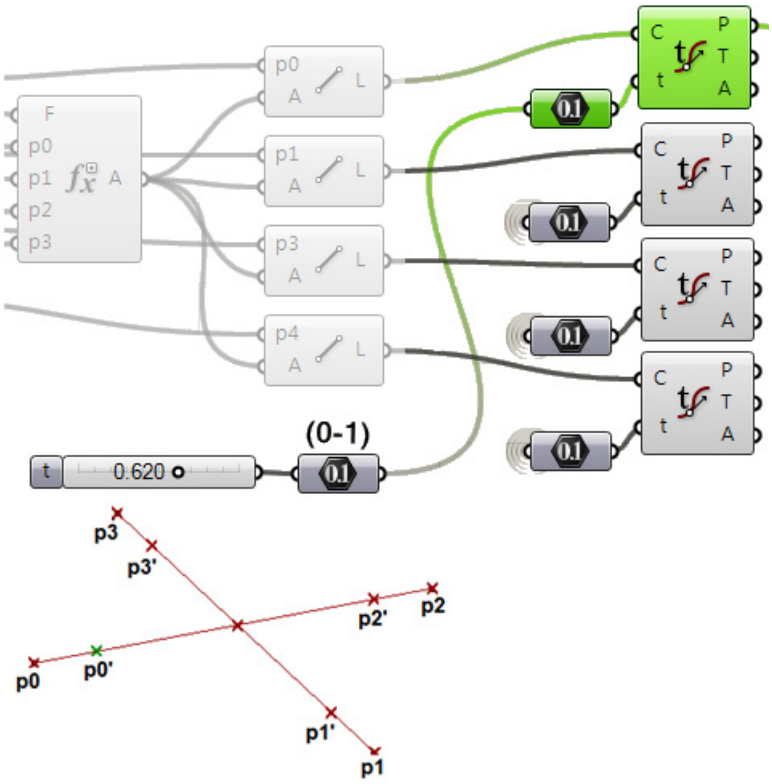


The component logic defines four corner points and a midpoint, and then connects each of the corner points with the center using the Grasshopper ***Line*** component.

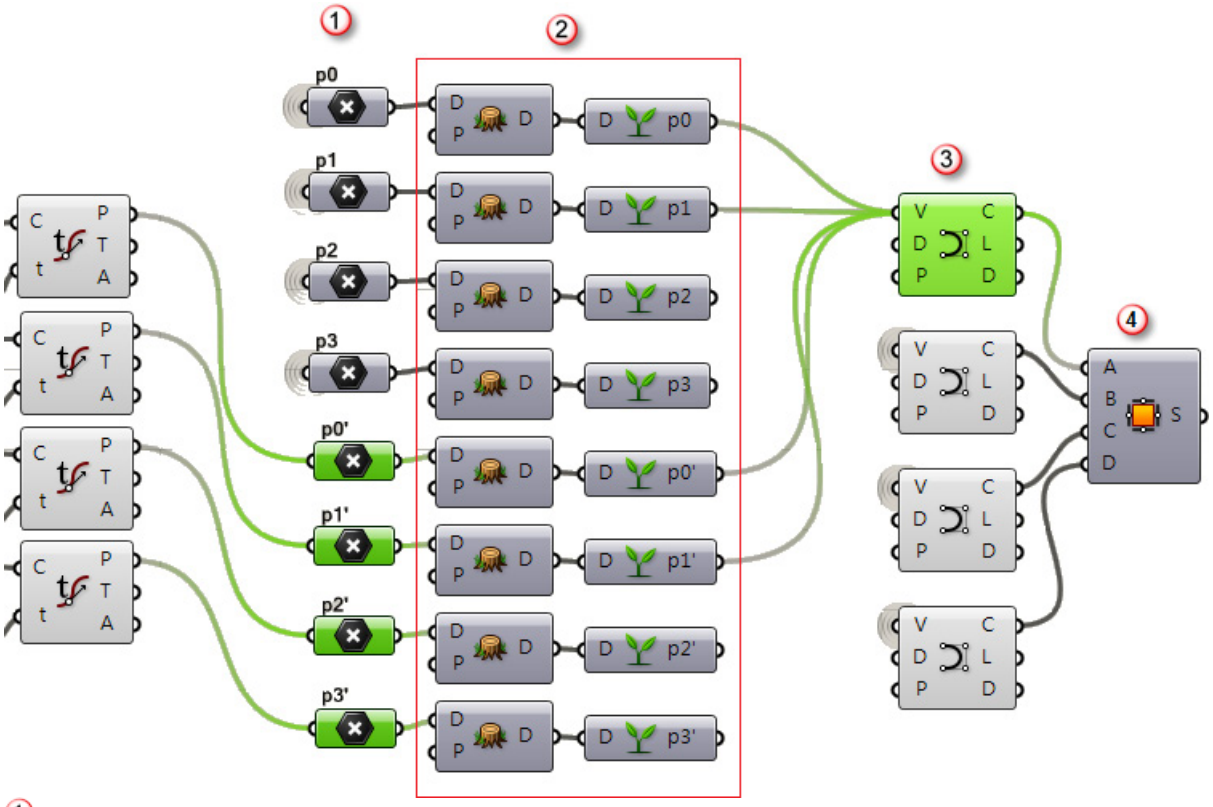


2. Define points on each of the lines that fall within the line's domain.

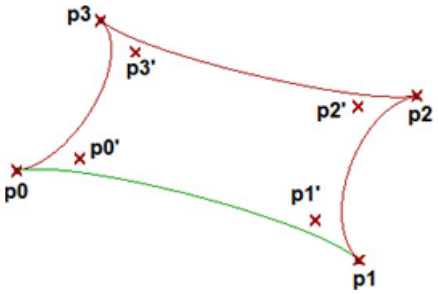
Here we are using a ***Number*** component to generate a constant number. You can also hook up a ***Number Slider*** with values between 0 and 1 to see the effect of changing the weight on the final component.



3. Create a curve using the corner and line points as illustrated.



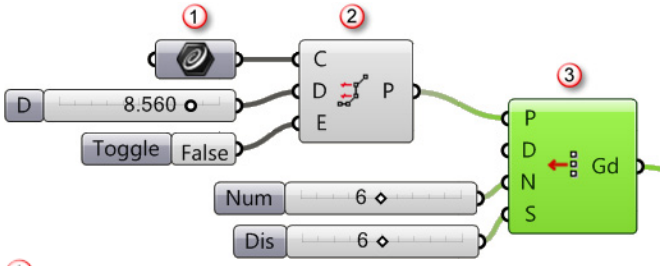
1 *All 8 component points*
2 *Flatten and graft necessary when hook to system*
3 *Create edge curves*
4 *Create edge surface*



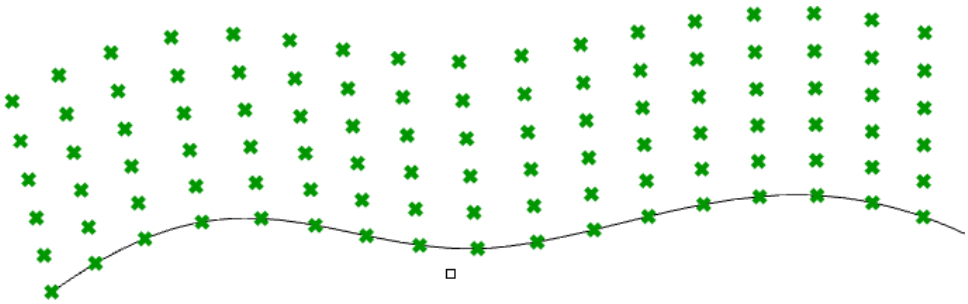
This concludes the creation of the parametric component. Each component represents a cell in the wall system that we will build next.

4. Create the wall system based on a curve created in Rhino and referenced in Grasshopper.

You can use any of the ***Curve Divide*** components to get the list of points to feed into the ***Extrude*** components.

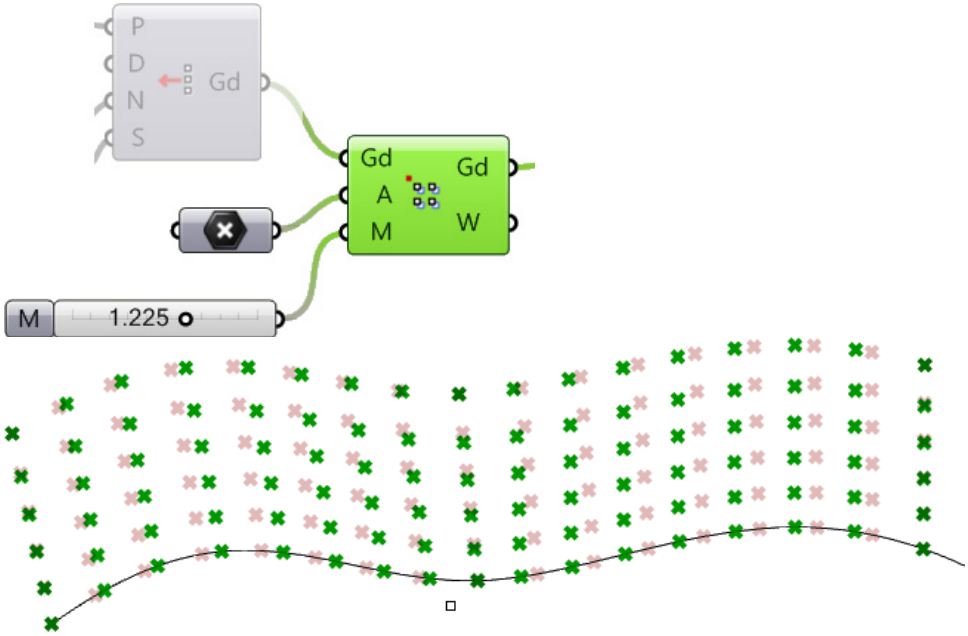


1 *Input base curve*
2 *Divide curves by distance*
3 *Grid from extrude*



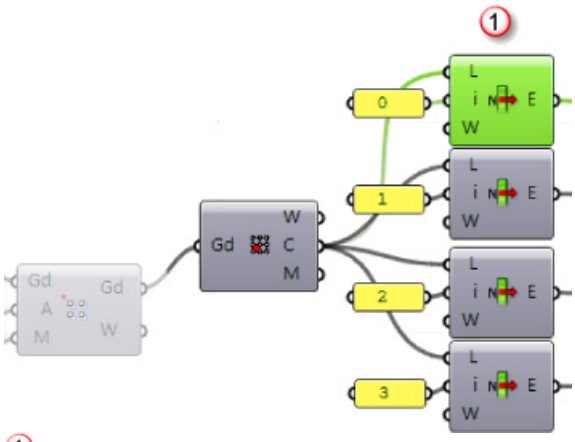
5. Use a ***PanelingTools > Grid Attractors > Point Attraction*** component to vary the cell size and shuffle the grid points.

   The green points shift towards the attractor point (red points are the original grid points locations).

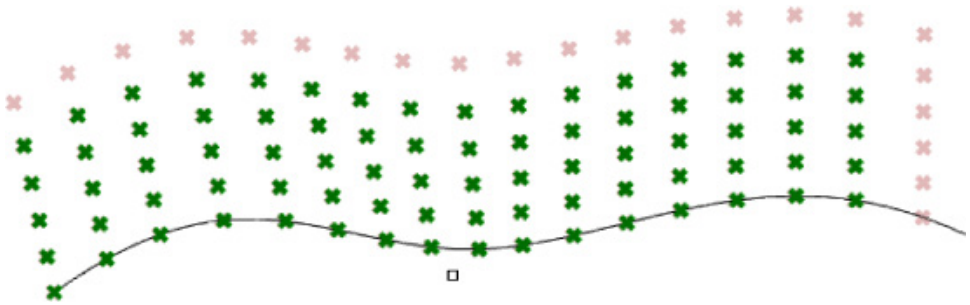


6. Use the ***PanelingTools > Panel2D > Cellulite*** component to extract lists of corner points of the system cells to feed into our component logic

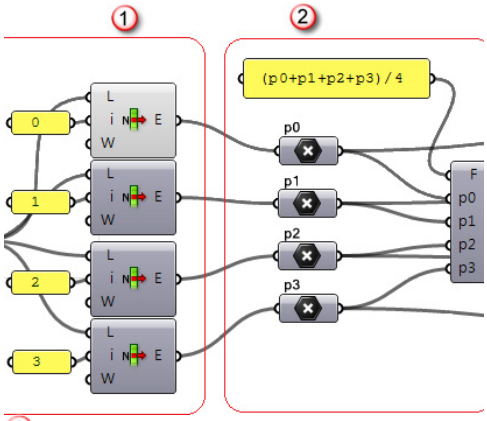
   Separate each of the corners as illustrated.



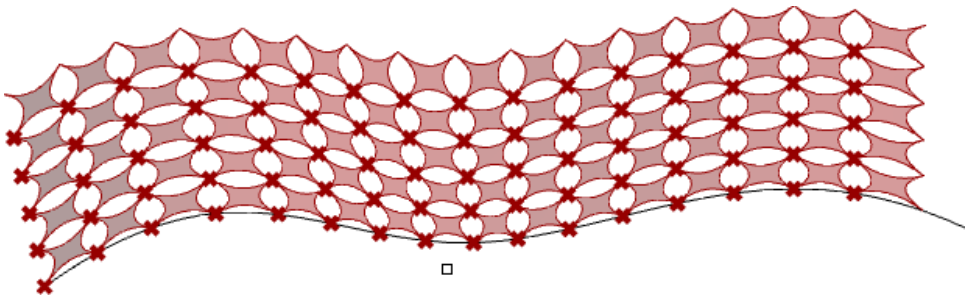
1 *Separate corners into 4 lists*



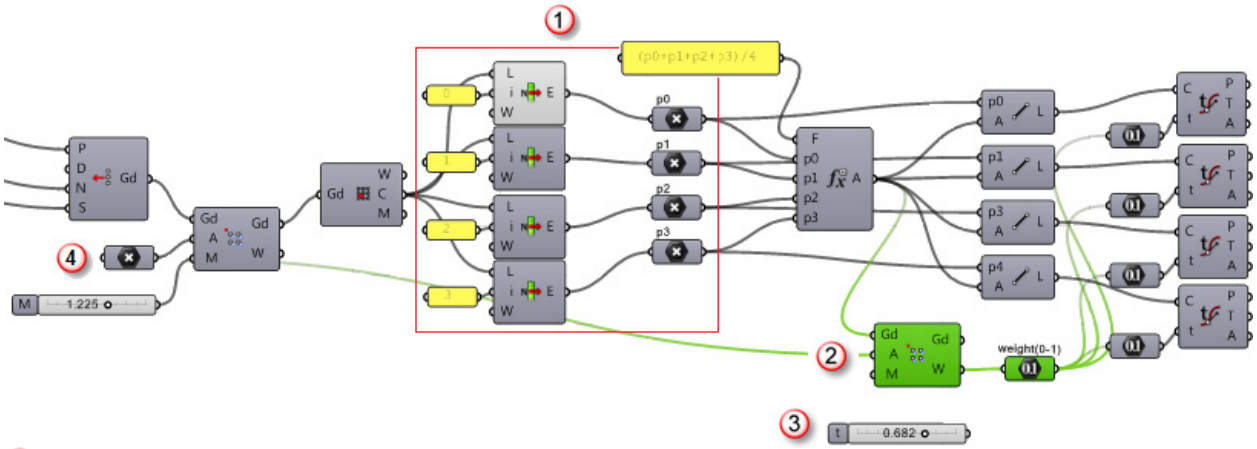
Once we hook the system into the component logic, we get components populated over the system.



1 *System logic*
2 *Component logic*

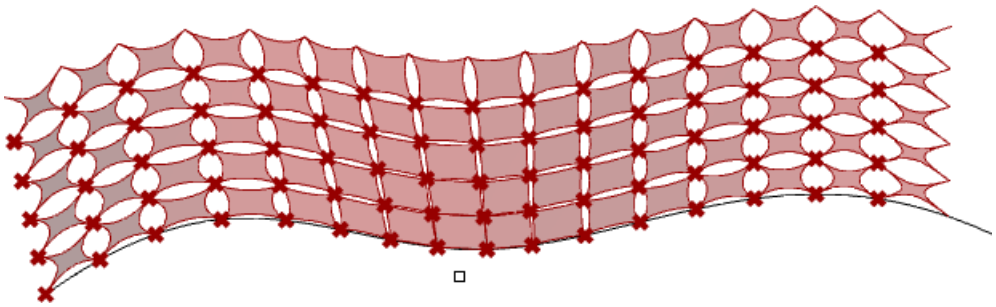


7. In order to make the component distribution variable, feed variable weights based on the distance from the attractor point.

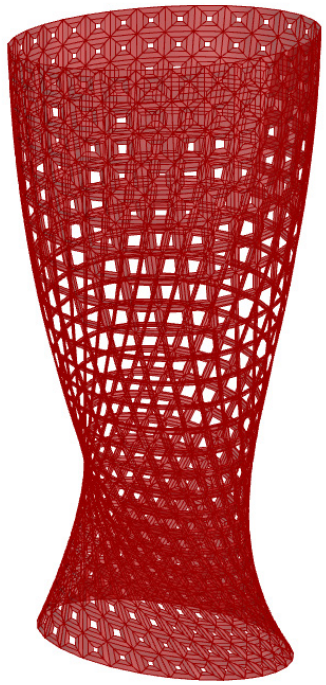


1. *Connect the system logic to the component logic*
2. *Variable weights*
3. *Constant weights*
4. *Attractor point*

Here is the result when using variable weights.



Tower



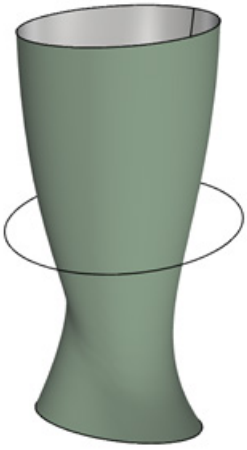
This tutorial distributes a list of variable mesh-based modules on the tower skin based on a curve attractor.

The modules have variable opening size and are distributed so that modules with the biggest opening attract closer to the curve. The modules were modeled in Rhino, but you can choose to parametrically define a module inside grasshopper and control the aperture using the attractors.

The first step is to module the geometry elements in Rhino. In this case, we have the tower surface, attractor curve and the module-list.

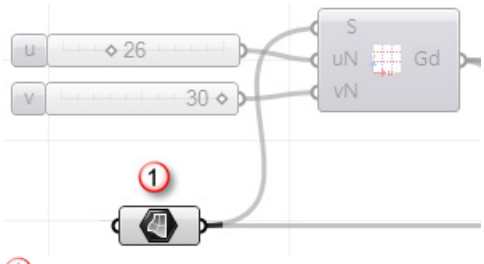
To see an example

- Open ***..\PT_GH_PrimerExamples\Tutorial Examples\Tower.3dm and Tower.gh***.



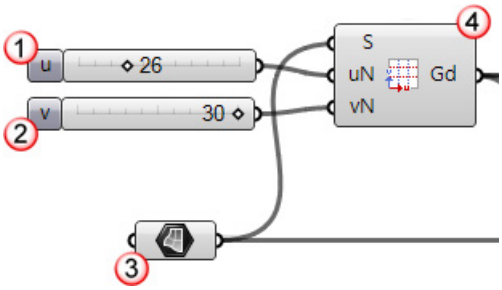
Create the tower panel

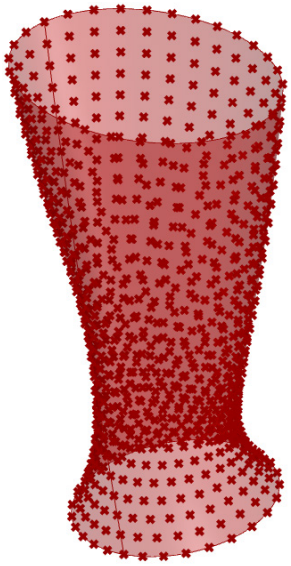
1. Start the definition with a ***Params > Geometry > Surface*** component and select the tower surface from Rhino.



1 *Reference surface parameter*

2. Feed the surface into a ***PanelingTools > Grid > Surface Domain Number*** component.



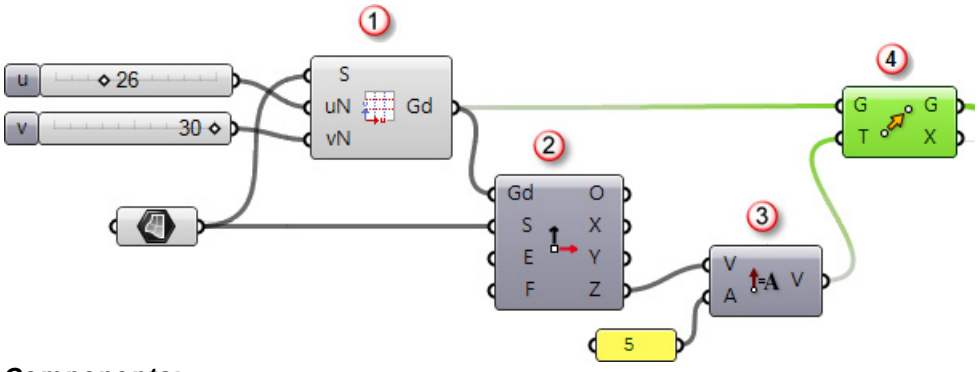


Components:

1 *Number Slider for the u-direction*
2 *Number Slider for the v-direction*
3 *Surface – input existing surface*
4 *Grid > Surface Domain Number*

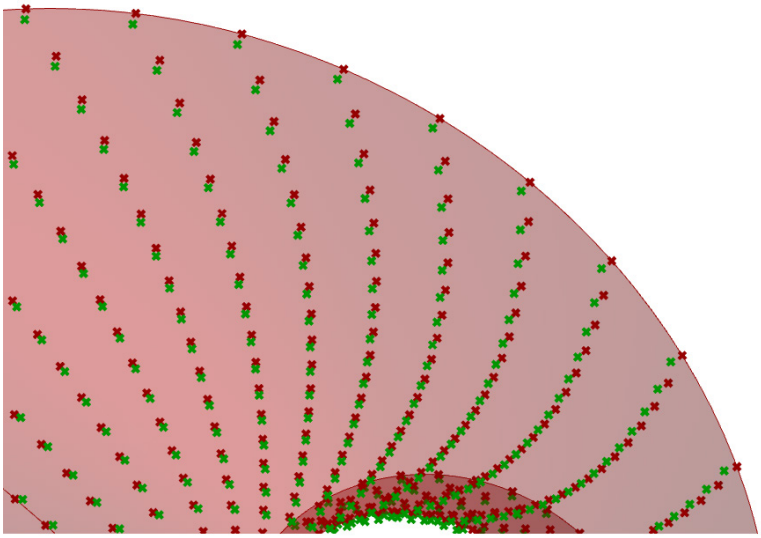
Once we have the grid, we need to offset in a direction normal to the surface.

3. Use the ***Coordinates*** component to calculate the normal direction at each grid point.

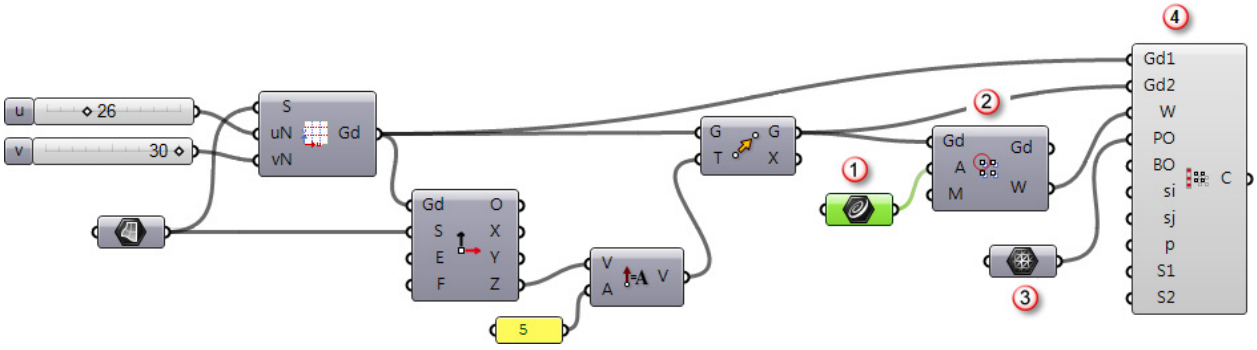


Components:

1. *Input grid*
2. *Coordinate – calculates the x, y, z direction of each grid point relative to input surface*
3. *Amplitude – specifies the offset amount*
4. *Move – moves the grid point in normal direction*



4. Create an attraction field (grid of weights) to feed into the paneling component.



Components:

1. *CrvAtts - attractor curve*
2. *Curve Attraction - calculates weights at each grid point*
3. *Mesh - input list of modules*
4. *Morph3D List - distributes the list of components on the grid using attraction values*

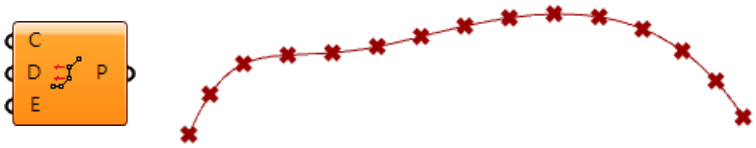
Section II: Components Reference

Chapter 3: Curve Components

These components divide a NURBS curve using various controls.

Divide Distance (ptDivideDis)

The ***Divide Distance (ptDivideDis)*** component calculates divide points on a curve or list of curves by specified distance with an option to add end point(s).



Input

C: Curve(s) to divide

D: Distance between points

E: Option to add end point if set to *True*

Output

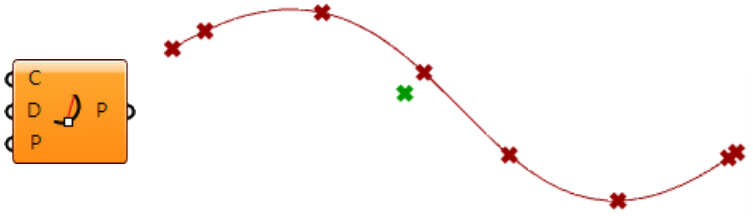
P: List of divide points

Example

- Open ***..\PT_GH_PrimerExamples\Component Examples\ptDivideDis.gh***.

Divide Distance with Reference (ptDivideDisRef)

The ***Divide Distance with Reference (ptDivideDisRef)*** component divides curves by distance with reference point. Reference point is used to control the location of divide points.



Input

C: Curve(s) to divide

D: Distance between points

P: Reference point. Calculates the closest point on the curve to the reference point, then divides the two sides of the curve by the distance *(D)*

Output

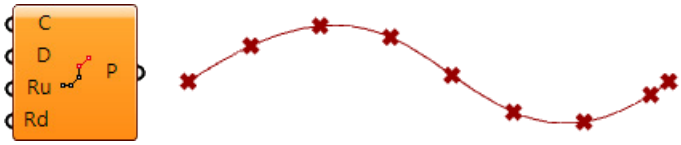
P: List of divide points

Example

- Open ***..\PT_GH_PrimerExamples\Component Examples\ptDivideDisRef.gh***.

Divide Length (ptDivideLen)

The ***Divide Length with Reference (ptDivideLen)*** component calculates divide points on a curve or list of curves by specified length on the curve with an option to round the distance up or down. If rounded, the curve is divided into equal lengths.



Input

C: Curve(s) to divide

D: Length on curve

Ru: Round the length up if set to *True*

Rd: Round the length down if set to *True*

Output

P: List of divide points

Example

- Open ***..\PT_GH_PrimerExamples\Component Examples\ptDivideLen.gh***.

Divide Length with Reference (ptDivideLenRef)

The ***Divide Length with Reference (ptDivideLenRef)*** component calculates divide points on a curve or list of curves by specified length on the curve. A reference point is used to control the location of the divide points.

Input

C: Curve(s) to divide

D: Length on curve

P: Reference point. Calculates the closest point on the curve to the reference point, and then divides the two sides of the curve by the distance *(D)*

Output

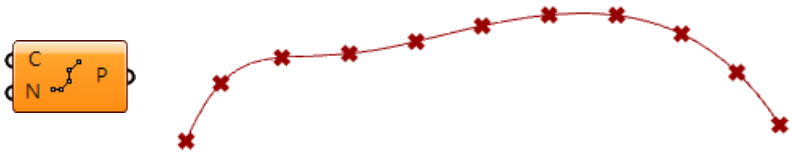
P: List of divide points

Example

- Open ***..\PT_GH_PrimerExamples\Component Examples\ptDivideDisRef.gh***.

Divide Number (ptDivideNum)

The ***Divide Number (ptDivideNum)*** component calculates divide points on a curve or list of curves by a specified number.



Input

C: Curve(s) to divide

N: Number of spans. An input of 10 generates 11 points

Output

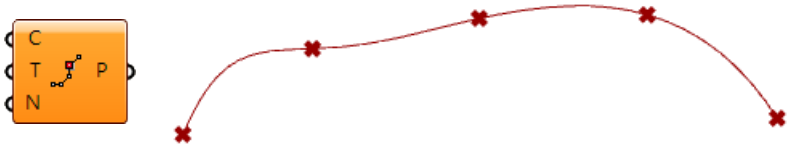
P: List of divide points

Example

- Open ***..\PT_GH_PrimerExamples\Component Examples\ptDivideNum.gh***.

Divide Parameter (ptDivideParam)

The ***Divide Parameter (ptDivideParam)*** component calculates divide points on a curve f list of curves from a list of parameters.



Input

C: Curve(s) to divide

T: Parameter list

N: Normalize if set to *True*

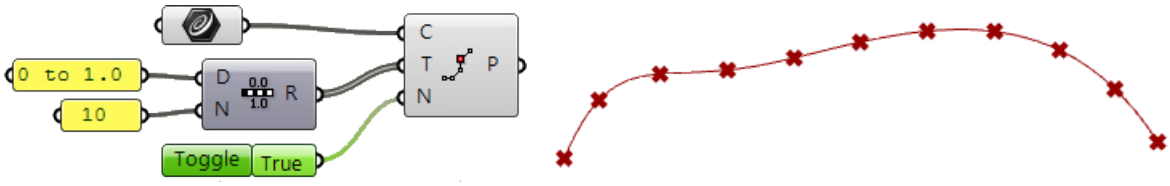
Output

P: List of divide points.

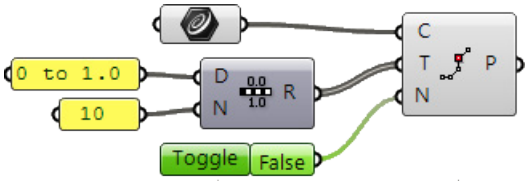
Example

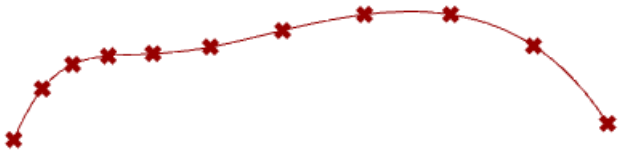
- Open ***..\PT_GH_PrimerExamples\Component Examples\ptDivideParam.gh***.

When *Normalize (N)* is set to *True*, the input *Curve (C)* is divided to equal spans if the distance between parameter values is equal, regardless of the curve parameterization or domain.



When Normalize *(N)* is set to false, the input *Curve (C)* is divided into equal spans in parameter space, which may not translate to equal spans in 3-D space.





Chapter 4: Grid Components

These components generate paneling grids and organize them in a tree structure where branches represent rows of points.

Compose Grid (ptCompose)

The ***Compose Grid (ptCompose)*** component creates grids from scratch. It takes a list of points and two integer lists to define the (i,j) location of each point in the grid

Input

P: List of points

i: List of i indices of corresponding points

j: List of j indices of corresponding points

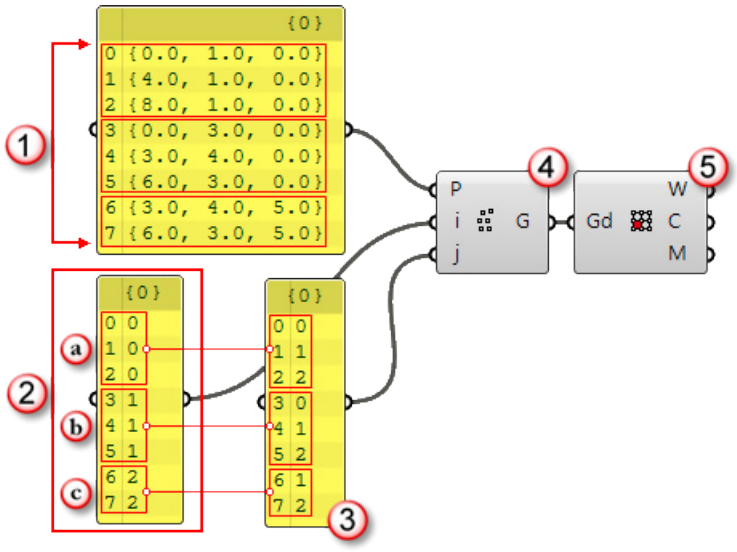
Output

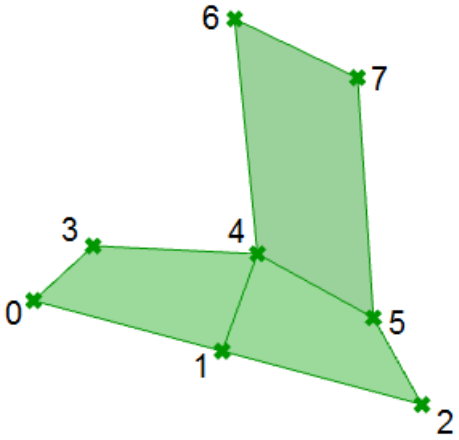
Gd: Grid

Example

Open ***..\PT_GH_PrimerExamples\Component Examples\ptCompose.gh***.

The example shows how to organize a list of eight points into a grid. Define the index of each point in the grid (row, column location). For example, the first three points make the first row in the grid. Notice that their row location *(i)* is set to 0, and the column locations *(j)* are 0, 1, and 2.





Components:

1. *Panel - list of points*
2. *Panel - row indices of the points*
   - a. *Panel - first row*
   - b. *Panel - second row*
   - c. *Panel - third row*
3. *Panel - column indices of the points*
4. *Compose - creates the grid*
5. *Cellulate - generates cell data*

Compose Grid Number (ptComposeNum)

The ***Compose Grid Number (ptComposeNum)*** component assumes that the resulting grid will have equal number of points in each row (rectangular grid). It takes a list of points and number of rows and outputs the grid. If the number of input points is not divisible by the input number of rows, then the remainder will make the last row of the grid.

Input

P: List of points

N: Number of rows

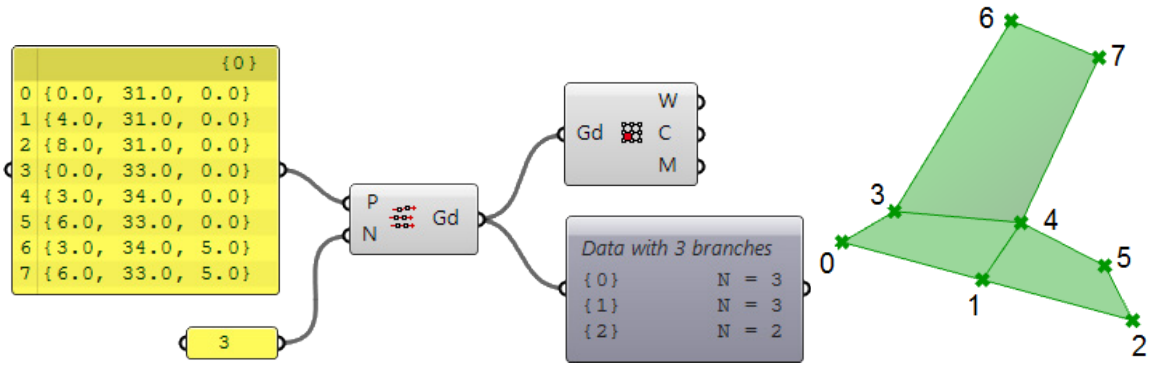
Output

Gd: Grid

Example

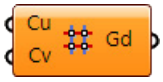
Open ***..\PT_GH_PrimerExamples\Component Examples\ptComposeNum.gh***.

The example shows how to organize a list of eight points into a grid that has three rows. It assumes that points are listed in order. This means that the first input point becomes the base point (the first point of the first row). The second input point becomes the second point in the first row, and so on until all rows are filled. Note that the last row contains fewer points.



Intersect Curves (ptUVCrvs)

The ***Intersect Curves (ptUVCrvs)*** component generates a grid from intersecting two sets of curves. The first set defines the row direction of the grid, and the second set defines the column direction.



Input

Cu: List of curves in the u-direction (row)

Cv: List of curves in the v-direction (column)

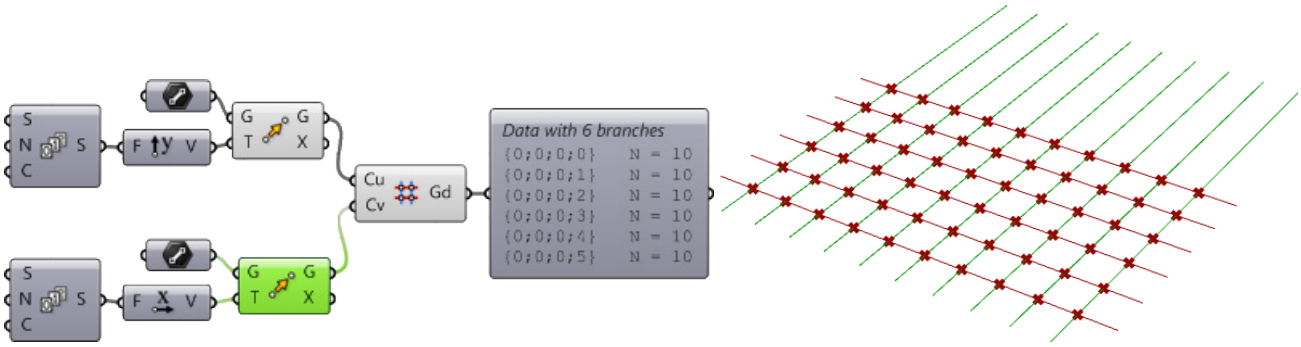
Output

Gd: Grid

Example

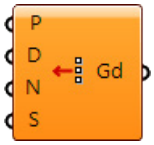
- Open ***..\PT_GH_PrimerExamples\Component Examples\ptUVCurves.gh***.

In the example, two sets of lines in the x- and y-directions are generated, and the grid is created from the intersections of the lines. The generated grid has six rows and ten columns. Intersecting curves do not have to be planar or linear.



Planar Extrude (ptPlanarExtrude)

The ***Planar Extrude (ptPlanarExtrude)*** component creates extruded grids.



Input

P: List of points (represents first row)

D: Extrude direction (represents columns direction)

N: Number of rows

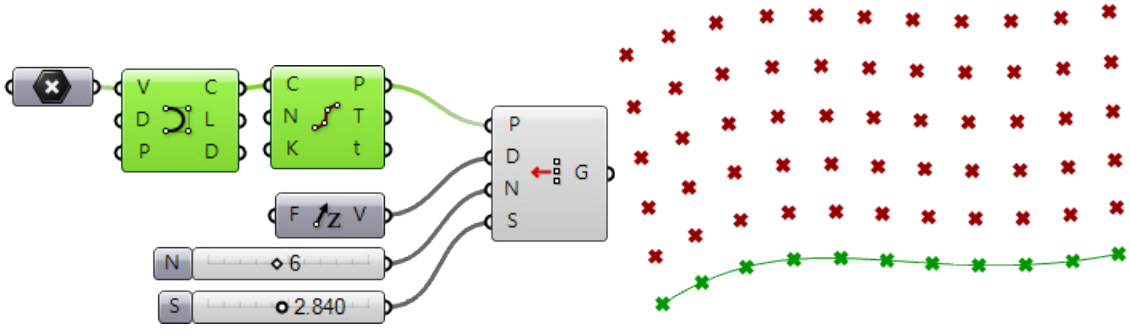
S: Distance between rows

Output

Gd: Grid

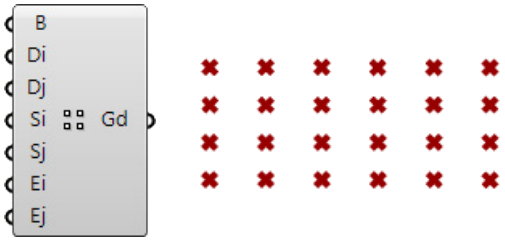
Example

- Open ***..\PT_GH_PrimerExamples\Component Examples\ptPlanarExtrude.gh***.



Planar Grid (ptPlanar)

The ***Planar Grid (ptPlanar)*** component creates parallel planar grids. The u- and v-directions of the grid do not have to be orthogonal.



Input

B: Base point

Di: Direction of the rows

Dj: Direction of the columns

Si: Distance between rows

Sj: Distance between columns

Ei: Number of rows

Ej: Number of columns

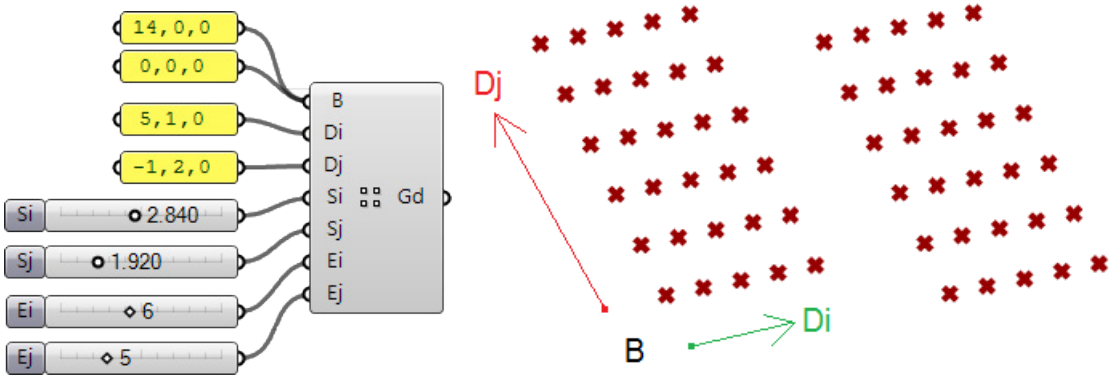
Output

Gd: Grid

Example

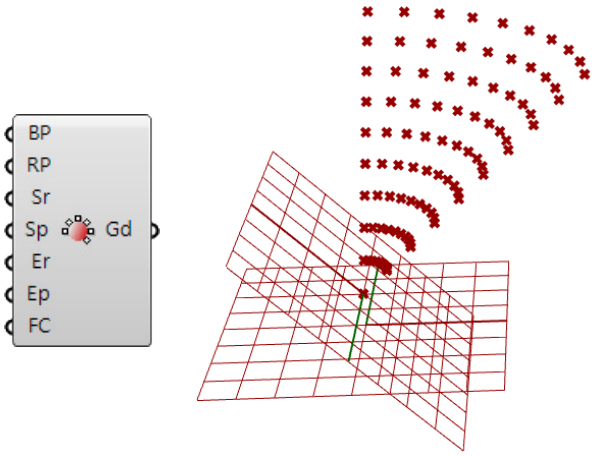
Open ***..\PT_GH_PrimerExamples\Component Examples\ptPlanar.gh***.

Planar grids can have specified row and column directions. Using multiple base points creates more than one grid represented in one tree structure.



Polar 3D Grid (ptPolar3D)

The ***Polar 3D Grid (ptPolar3D)*** component creates polar 3-D grids. The component takes two input planes. The first defines the base plane and rotation axis, and the second defines the grid base point and the first row direction.



Input

BP: Base plane - defines the base plane and rotation axis

RP: Revolve plane - defines the grid base point and the first row direction

Sr: Distance between points in the radial direction

Sp: Angle (in degrees) in the polar direction

Er: Number of points in the radial direction (number of columns)

Ep: Number of points in the polar direction (number of rows)

FC: Create a full closed circle (ignores angle set in *Sp*)

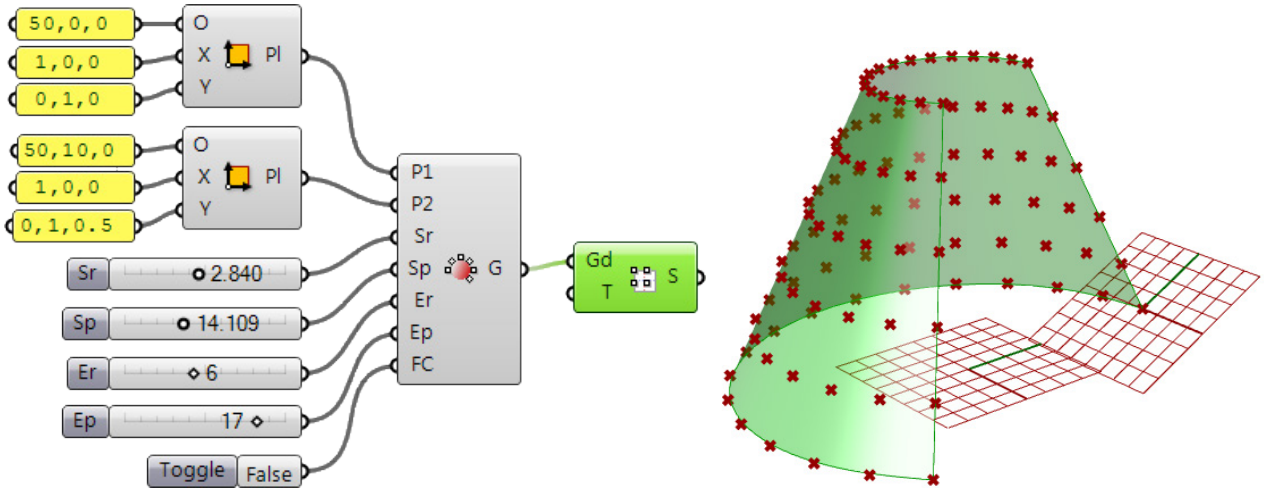
Output

Gd: Grid

Example

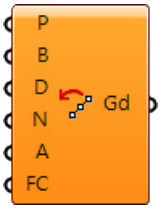
- Open ***..\PT_GH_PrimerExamples\Component Examples\ptPolar3D.gh***.

You can define plane direction and input multiple planes.



Polar Extrude (ptPolarExtrude)

The ***Polar Extrude (ptPolarExtrude)*** component creates polar 3-D grids. The component takes two input planes: the base plane and rotation axis and the grid base point and the first row direction.



Input

P: List of points (represent first row)

B: Base point of the rotation axis

D: Rotation axis

N: Number of rows

A: Angle between rows

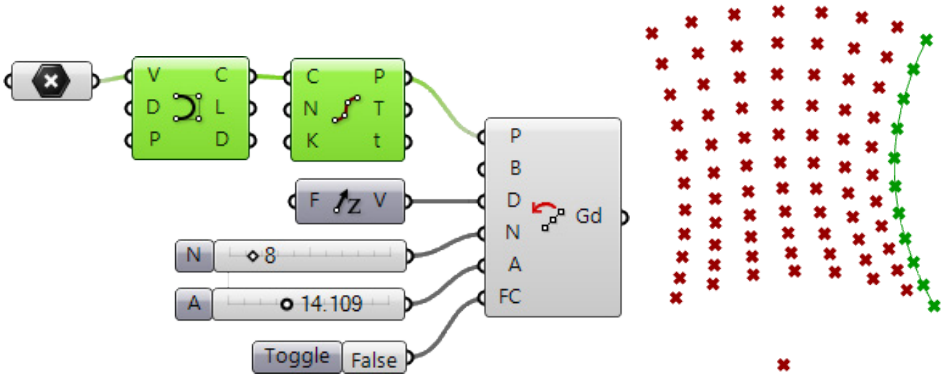
FC: Create a full closed circle (ignores angle set in A)

Output

Gd: Grid

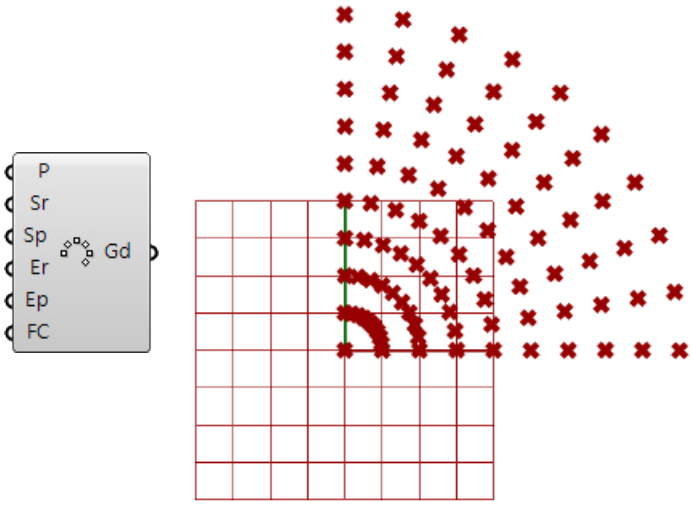
Example

- Open ***..\PT_GH_PrimerExamples\Component Examples\ptPolarExtrude.gh***.



Polar Grid (ptPolar2D)

The ***Polar Grid (ptPolar2D)*** component creates polar planar grids.



Input

B: Base plane
Sr: Distance between points in radial direction
Sp: Angle (in degrees) in polar direction
Er: Number of points in radial direction (number of columns)
Ep: Number of points in polar direction (number of rows)
FC: Create a full closed circle (ignores angle set in Sp)

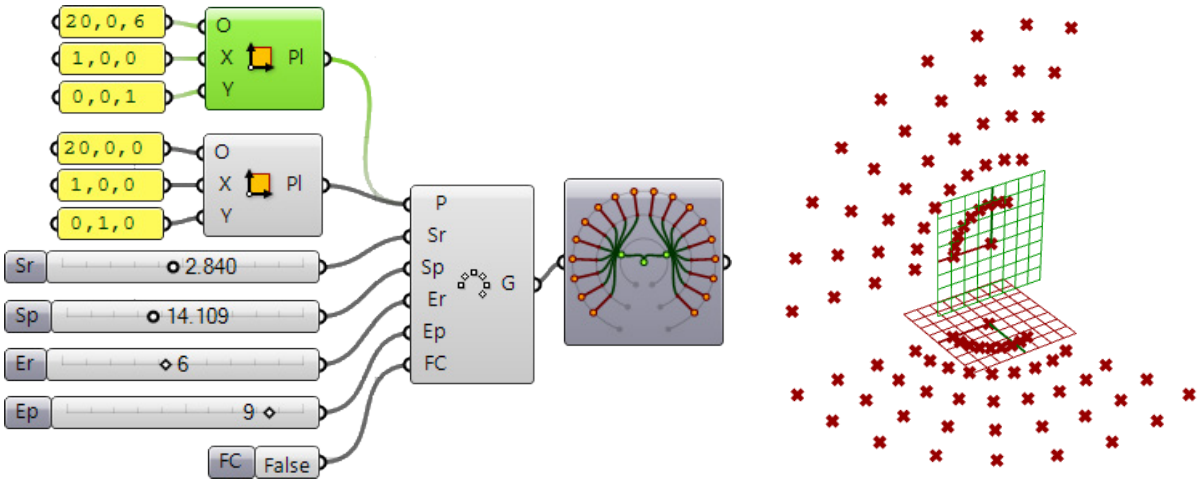
Output

Gd: Grid

Example

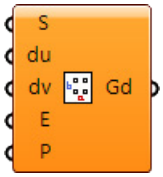
Open ***..\PT_GH_PrimerExamples\Component Examples\ptPolar2D.gh***.

You can define a plane direction and input multiple planes.



Surface Distance (ptSrfDis)

The ***Surface Distance (ptSrfDis)*** component attempts to divide a surface by equal distances. The underlying algorithm can take some time to calculate. The distances in u-and v-direction must be multiples of each other. The distance between points is maintained throughout the grid. In some cases, because of the surface curvature, the grid cannot cover the whole surface.



Input

S: Input surface

du: Distance in the u-direction

dv: Distance in the v-direction

E: Calculate on extended surface to achieve better coverage

P: (optional) Reference point - sets a uv parameter on the surface that the grid is forced through

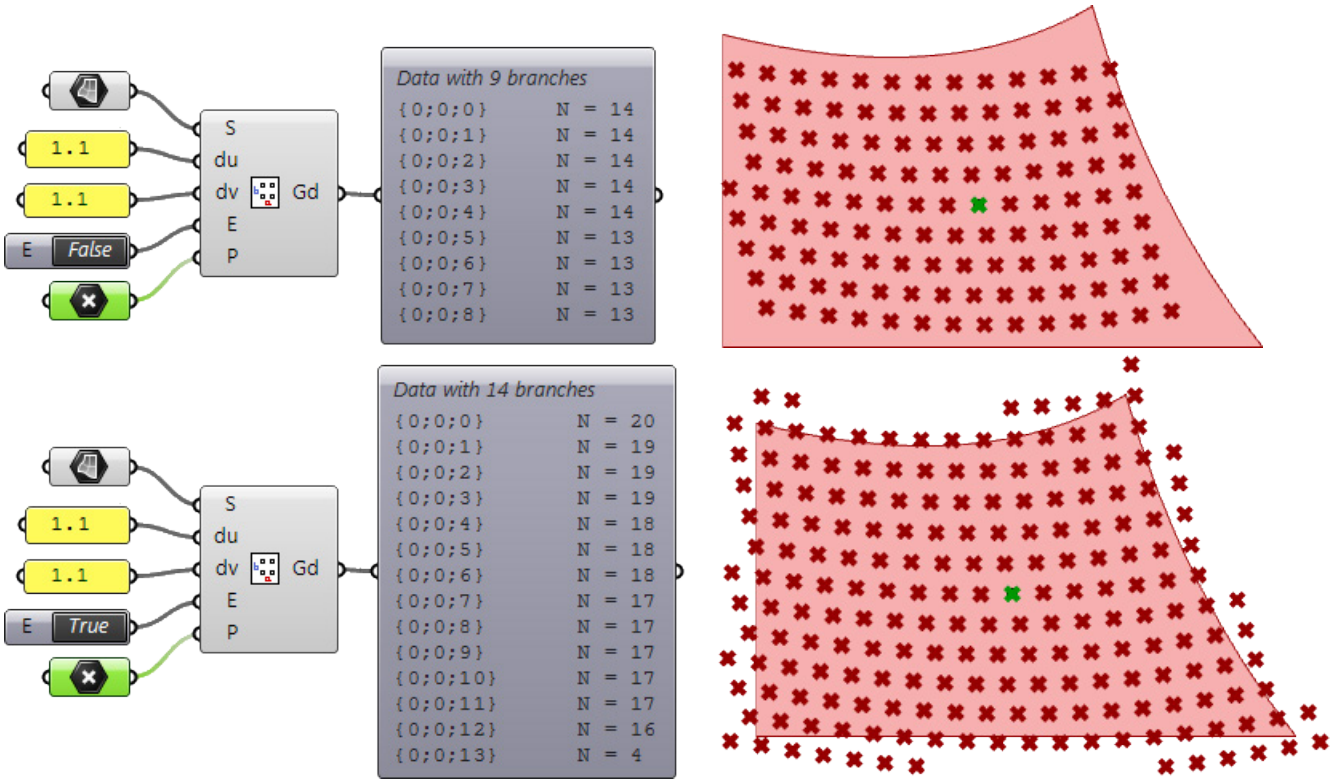
Output

Gd: grid

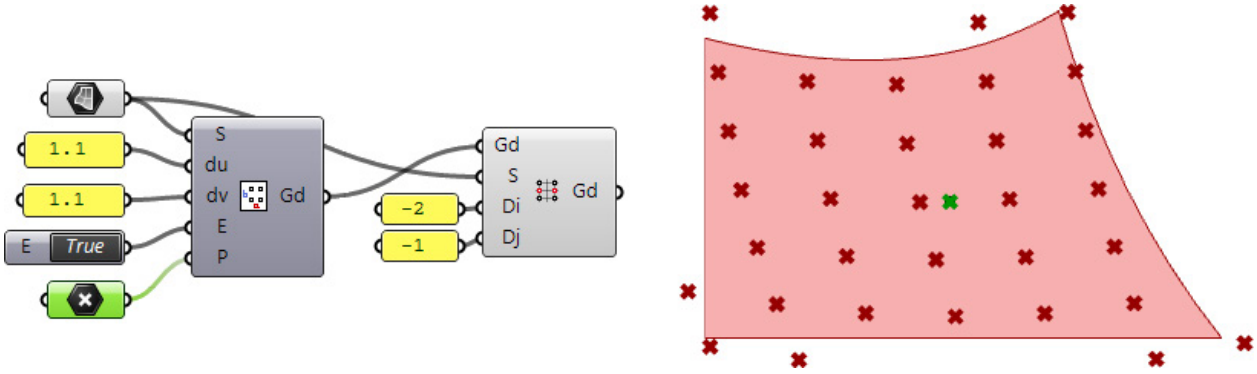
Example

Open ***..\PT_GH_PrimerExamples\Component Examples\ptSrfDis.gh***.

The following example shows the difference between setting *E* input parameter to *False* and *True*. The v-distance *(dv)* is a multiple of the u-distance *(du)*. If you need distances like 2.2 and 3.3, you can set the grid to be 2.2 and 1.1 and then decrease grid density (in the 1.1 direction) using the ***Grid Density*** component.

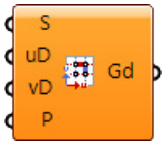


The v-distance *(dv)* has to be the same of a multiple of the u-distance *(du)*. If you need non-multiple distances, for example to be 3.3 in the u-direction and 2.2 in the v-direction, you can decrease the grid density using the ***Grid Density*** component as in the following.



Surface Domain Chord Distance (ptSrfDomChord)

The ***Surface Domain Chord Distance (ptSrfDomChord)*** component is similar to the ***Surface Domain Length*** component except the distance between points is set to the 3-D direct distance rather than the length on the curve. The ***Surface Domain Chord Distance*** component has similar options with the ability to set reference point to control the location of the grid.



Input

S: Input surface

uD: Distance in u-direction

vV: Distance in v-direction

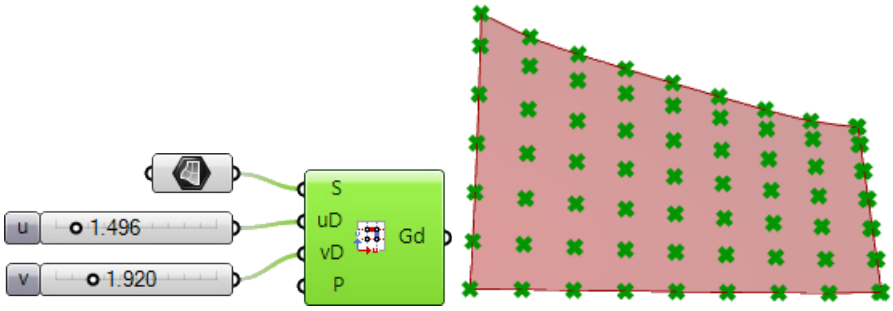
P: Reference point

Output

Gd: Grid

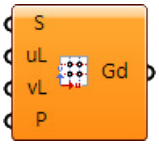
Example

- Open ***..\PT_GH_PrimerExamples\Component Examples\ptSrfDomChord.gh***.



Surface Domain Length (ptSrfDomLen)

The ***Surface Domain Length (ptSrfDomLen)*** component gives control over how to divide a surface domain using parameter space.



Input

S: Input surface

uL: Length on surface in u-direction

vL: Length on surface in v-direction

P: Reference point

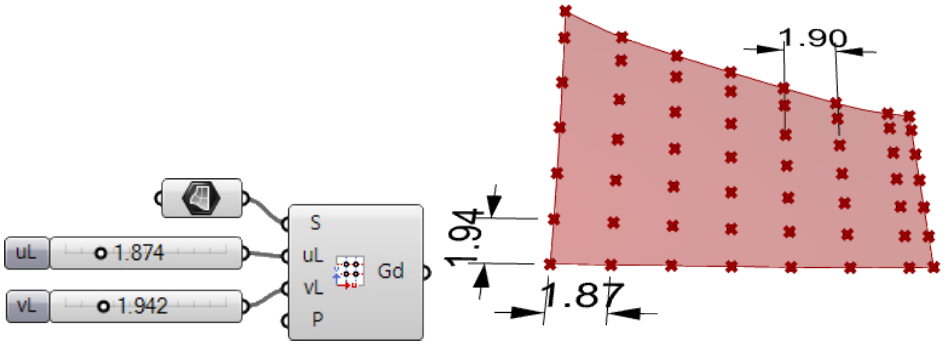
Output

Gd: Grid

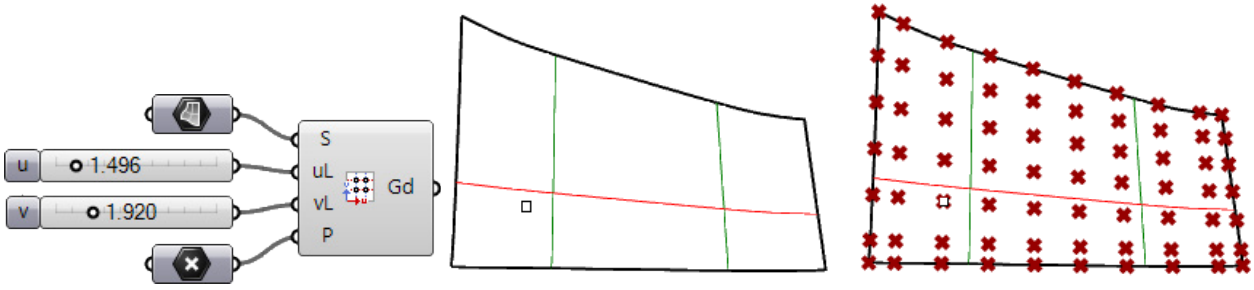
Example

Open ***..\PT_GH_PrimerExamples\Component Examples\ptSrfDomLen.gh***.

For a planar or extruded surface, the distances are exact across the surface, but if the surface is curved in two directions, distances will vary. The base u- and v-isocurves are divided exactly by the specified distance. The rest of the surface will probably have other distances depending on the shape of the surface. For free-form surfaces, it is impossible to follow the surface domain while maintaining constant distances on surface.

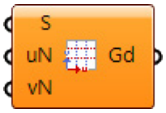


It is also possible to control the location of the grid on the surface by setting the *P* input parameter to a point on the surface. The exact divide distance is achieved on the isocurves that go through that reference point.



Surface Domain Number (ptSrfDomNum)

The ***Surface Domain Number (ptSrfDomNum)*** component generates a grid using a surface and follows the surface domain direction.



Input

S: List of curves in the u-direction (row)

uN: Number of spans in the u-direction

vN: Number of spans in the v-direction

Output

Gd: Grid

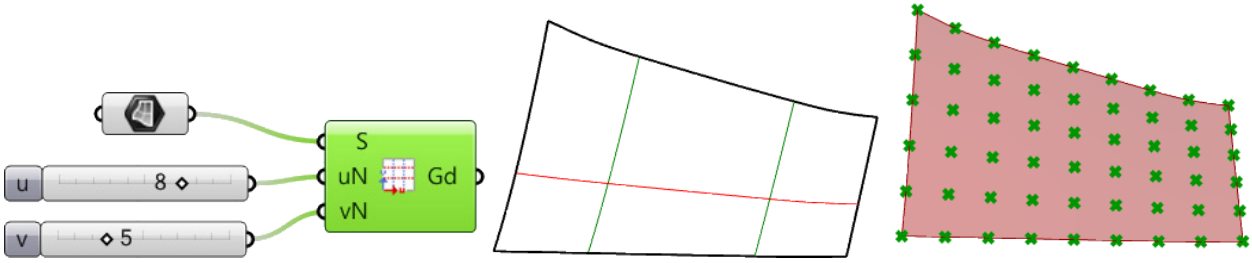
Example

- Open ***..\PT_GH_PrimerExamples\Component Examples\ptSrfDomNum.gh***.

The surface in the image below is divided into six branches; each has nine points.

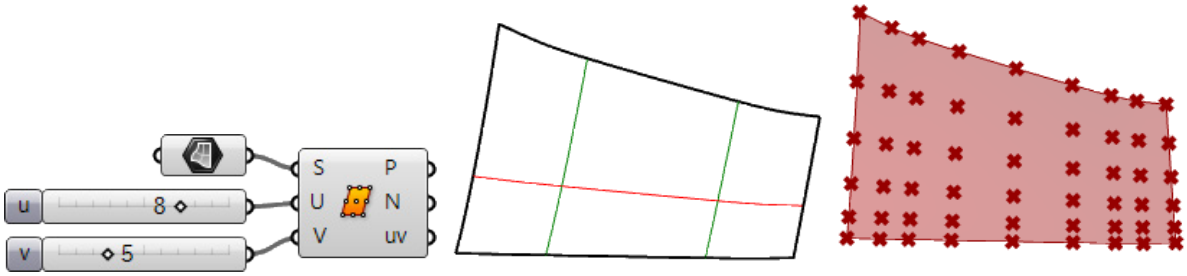
PanelingTools Surface Domain Number

When using ***Surface Domain Number***, while the points follow the isocurve directions of the surface, they are not affected by the surface parameterization. The surface will be divided equal distances if possible.



Contrast with Grasshopper Divide Surface

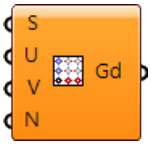
If you divide the same surface using Grasshopper ***Divide Surface*** component, the points might be spaced unevenly because the ***Divide Surface*** component actually divides the domain (in parameter space) into equal distances, which does not necessarily translate into equal spacing in 3-D space.



Another difference between the ***Surface Domain Number*** component and Grasshopper ***Divide Surface*** component is the way output is organized. The output tree from ***Surface Domain Number*** arranges rows data in branches, while ***Divide Surface*** arranges the data by columns.

Surface Parameter (ptSrfParam)

The ***Surface Parameter (ptSrfParam)*** component controls how to divide the surface domain using parameter space.



Input

S: Input surface

U: Parameter list (0-1) that divides the domain in u-direction

V: Parameter list (0-1) that divides the domain in v-direction

N: Option to use normalized distance to achieve more even distribution

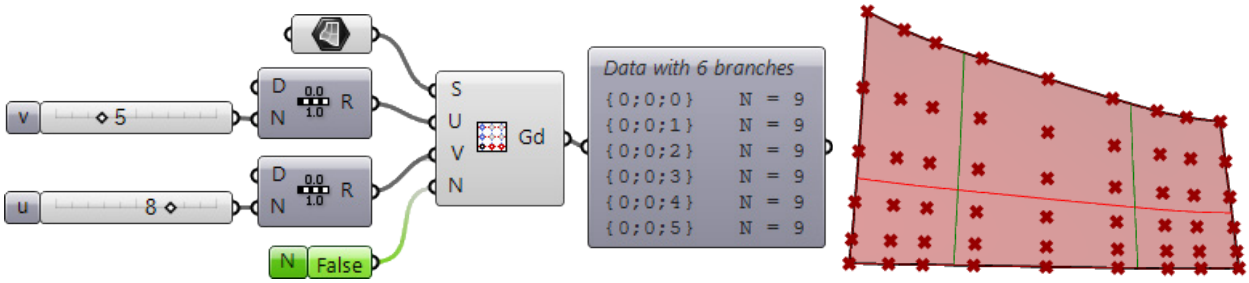
Output

Gd: Grid

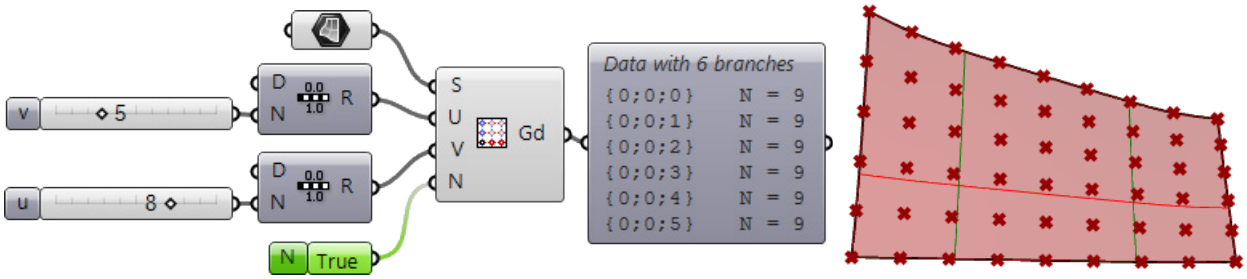
Example

Open ***..\PT_GH_PrimerExamples\Component Examples\ptSrfParam.gh***.

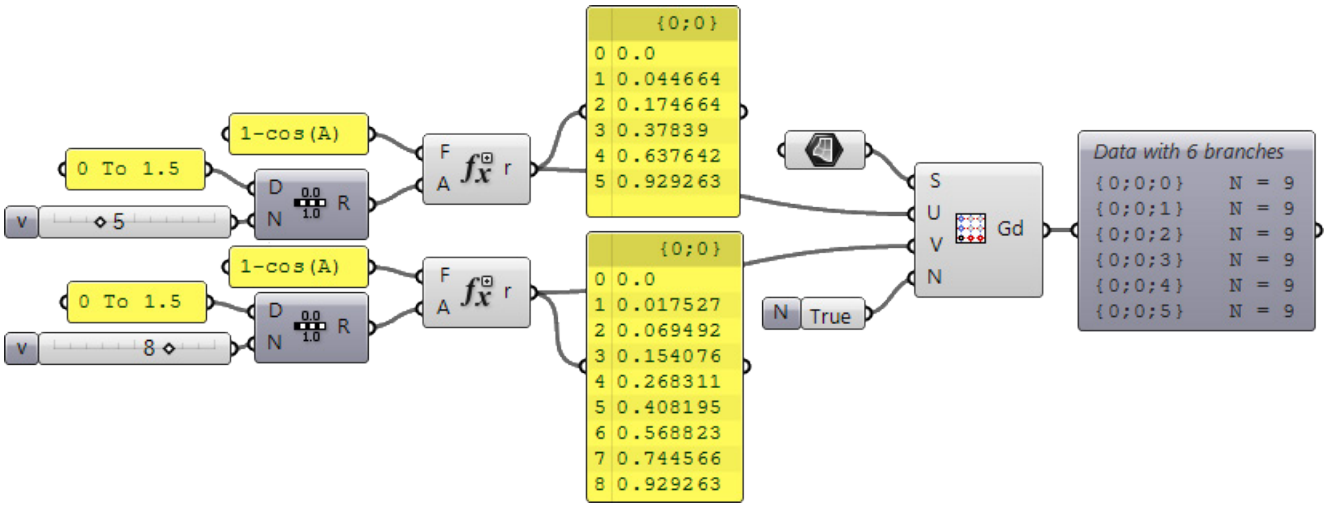
The ***Surface Parameter*** component controls how the domain is divided. Set *N* to *False* if you want the result to follow the surface parameterization. This achieves similar result to Grasshopper ***Divide Surface*** component, except that output data is organized with rows as branches, while ***Divide Surface*** organizes it with columns as branches.

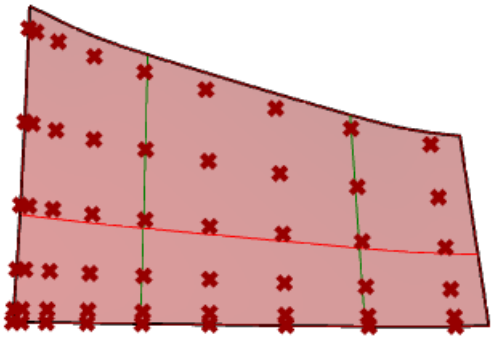


Set *N* to *True* to achieve a result similar to that of ***Surface Domain Number*** with relatively even distances.



This component is most useful when you have variable distances that you would like to divide. In this case, *N* to *True* to get an outcome that is not affected by how the input surface is parameterized, as shown in the next illustration.



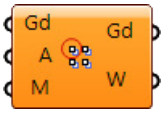


Chapter 5: Grid Attractors Components

These components adjust grid point locations based on various attractors.

Curve Attraction (ptCrvsAtts)

The ***Curve Attraction (ptCrvsAtts)*** component calculates new grid point locations based on attractor curve(s). It also calculates the corresponding grid of weights (influence). Attraction can be magnified by increasing the magnitude *(M)* value in the input. The edge points of the grid always move within the edge boundary keeping the outline of the grid intact.



Input

Gd: Input grid

A: Attractor curves

M: Magnitude or degree of influence. A negative input cause points to repel from the attractor

Output

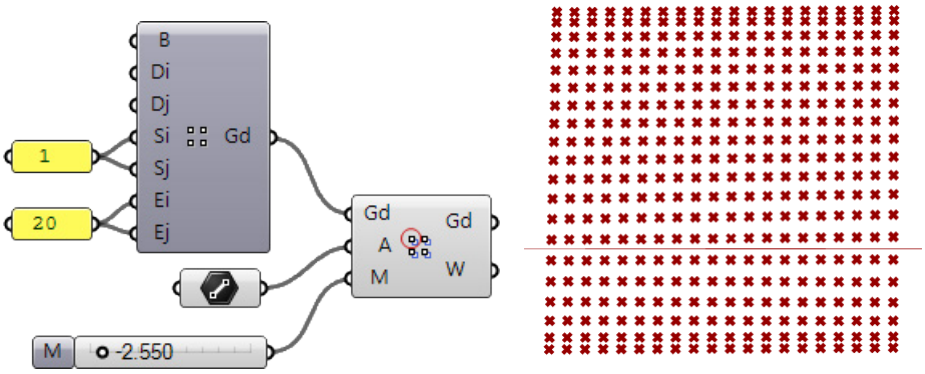
Gd: Resulting grid

W: Grid of weights (0–1)

Example

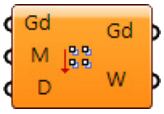
- Open ***..\PT_GH_PrimerExamples\Component Examples\ptCrvAtts.gh***.

The example generates a rectangular grid and then defines a centerline to be the attractor.



Direction Attraction (ptDirAtts)

The ***Direction Attraction (ptDirAtts)*** component calculates the attraction field based on relative angles between the normal direction of each point (on the grid surface) and that of a specified direction. This can be useful if, for example, you need to modify the grid based on the sun direction or the viewing angle.



Input

Gd: Input grid

M: Magnitude or degree of influence

D: Direction vector

Output

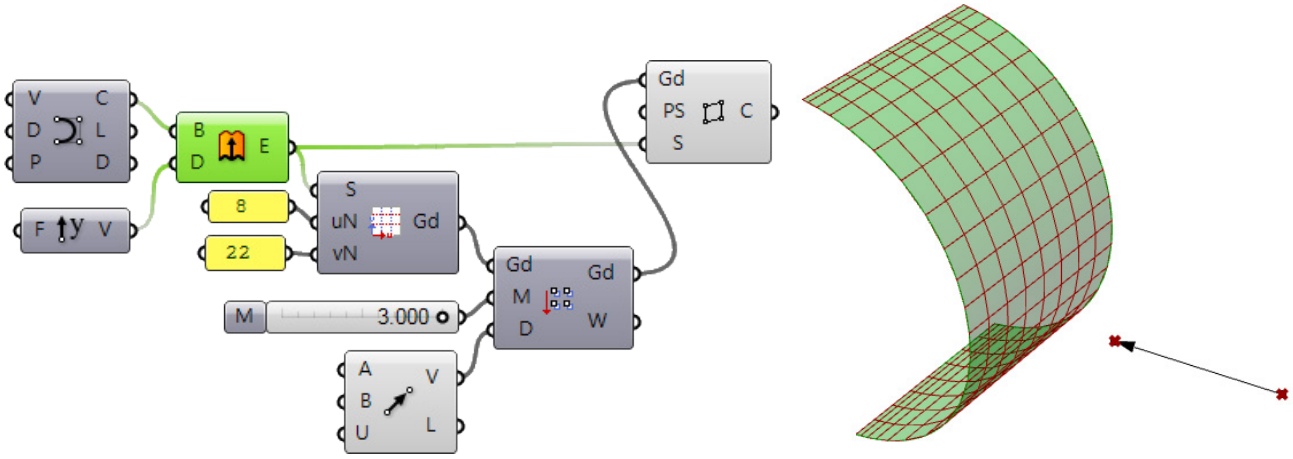
Gd: Resulting grid

W: Grid of weights (0-1)

Example

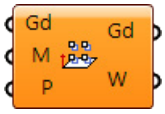
Open ***..\PT_GH_PrimerExamples\Component Examples\ptDirAtts.gh***.

The example shows how points are attracted using a direction curve. The definition first creates a surface and then divides it by a number to create the original grid. Then the ***Direction Attraction*** component creates denser spacing towards the edges. This is because the normal direction of the points at the edge forms a larger angle with the input direction compared to points towards the middle of the surface.



Draft Angle Attraction (ptSlopeAtts)

The ***Draft Angle Attraction (ptSlopeAtts)*** recalculates the input plane and the normal direction of each grid point relative to the grid surface. It also calculates the corresponding grid of weights (influence). Attraction can be magnified by increasing the Magnitude (*M*) value in the input. The edge points of the grid always move within edge boundary keeping the outline of the grid intact.



Input

Gd: Input grid

M: Magnitude or degree of influence

P: Input plane. By default uses world x-y plane

Output

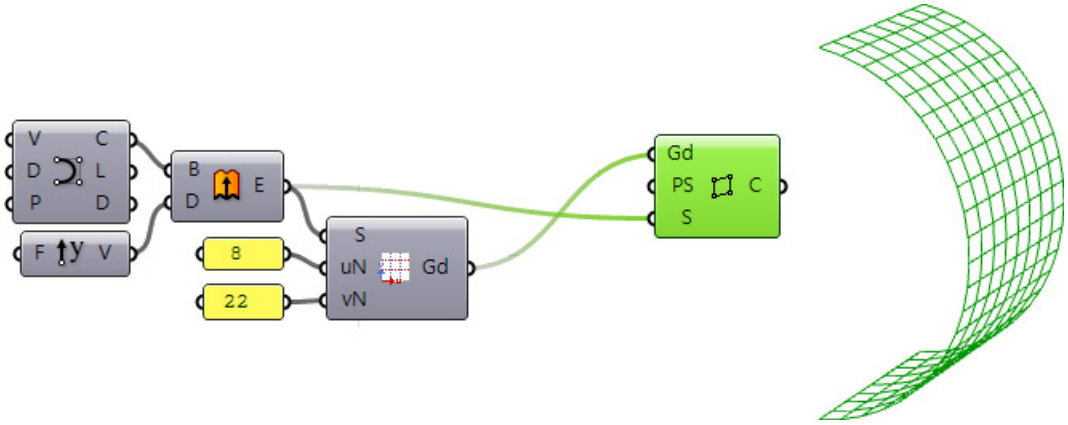
Gd: Resulting grid

W: Grid of weights (0-1).

Example

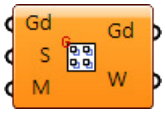
Open ***..\PT_GH_PrimerExamples\Component Examples\ptSlopeAtts.gh***.

The following example creates an evenly spaced grid using a surface, and then recalculates grid point locations using the slope attractor.



Gaussian Curvature (ptGauss)

The ***Gaussian Curvature (ptGauss)*** shuffles a grid of points on a surface using surface Gaussian curvature.



Input

Gd: Input grid

S: Surface

M: Magnitude or degree of influence

Output

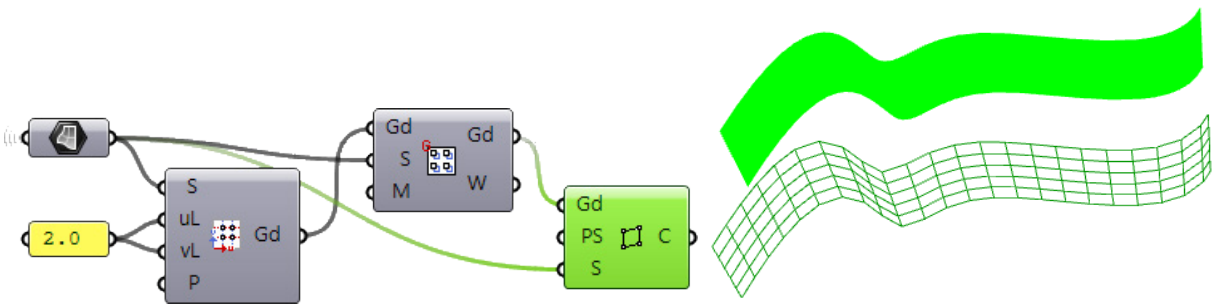
Gd: Resulting grid

W: Grid of weights (0-1)

Example

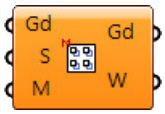
Open ***..\PT_GH_PrimerExamples\Component Examples\ptGauss.gh***.

Using the same surface as the previous section, the analysis graph shows us a uniform curvature; therefore, the attraction by Gaussian does not change the grid.



Mean Curvature (ptMean)

The ***Mean Curvature (ptMean)*** component shuffles a grid of points on a surface using surface mean curvature.



Input

Gd: Input grid

S: Surface

M: Magnitude or degree of influence

Output

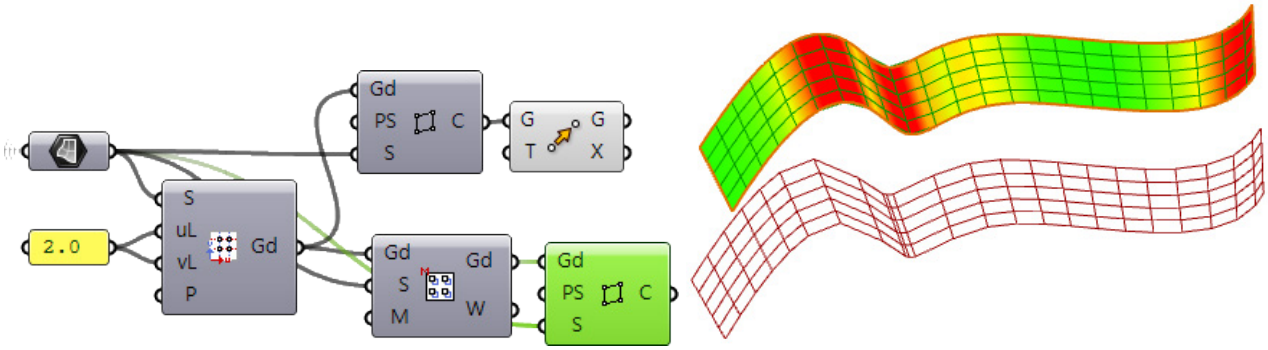
Gd: Resulting grid

W: Grid of weights (0-1)

Example

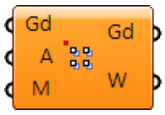
- Open ***..\PT_GH_PrimerExamples\Component Examples\ptMean.gh***.

In the top image, the surface is divided evenly using the underlying surface domain. The bottom part of the definition uses ***Mean Curvature*** to attract points towards the area of highest curvature.



Point Attraction (ptPtsAtts)

The ***Point Attraction (ptPtsAtts)*** component calculates new point locations based on an attractor point or points. It also calculates the corresponding grid of weights (influence). Attraction can be magnified by increasing the magnitude *(M)* value in the input. The edge points of the grid always move within the edge boundary, keeping the outline of the grid intact.



Input

Gd: Input grid

A: Attractor points

M: Magnitude or degree of influence. A negative input cause points to repel from the attractor

Output

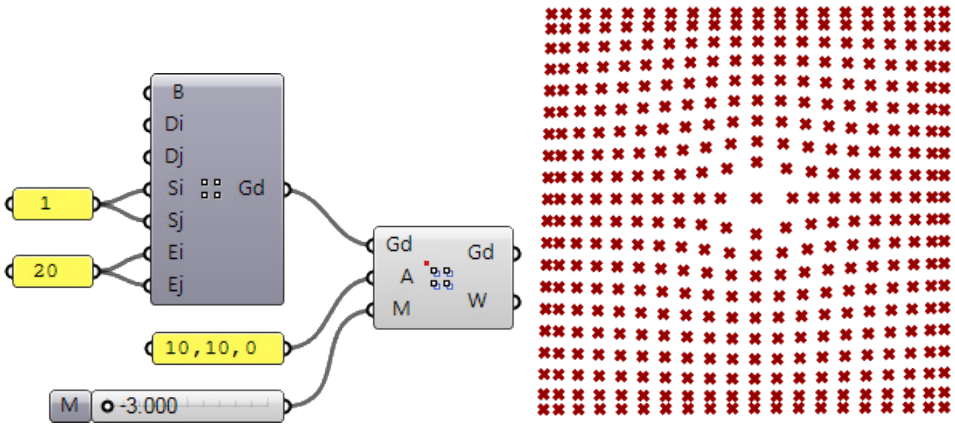
Gd: Resulting grid

W: Grid of weights (0-1)

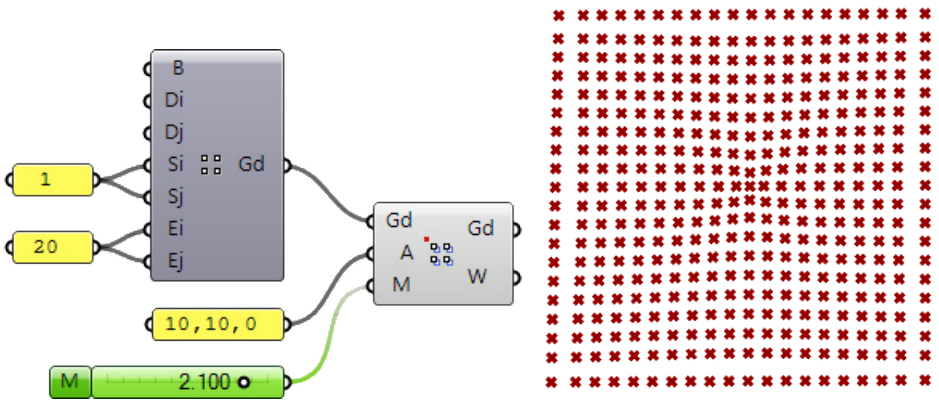
Example

Open ***..\PT_GH_PrimerExamples\Component Examples\ptPointAtts.gh***.

The following example generates a rectangular grid and then assigns a point in the middle of the grid to be the attractor. The magnitude *(M)* is set to a negative value, therefore the points are repelled from the center.

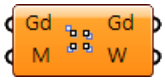


If magnitude *(M)* is set to a positive value, points are attracted towards the attractors.



Random Attraction (ptRandAtts)

The ***Random Attraction (ptRandAtts)*** component calculates new grid point locations randomly. Grid points will not collide or overlap.



Input

Gd: Input grid

M: Magnitude or degree of influence

Output

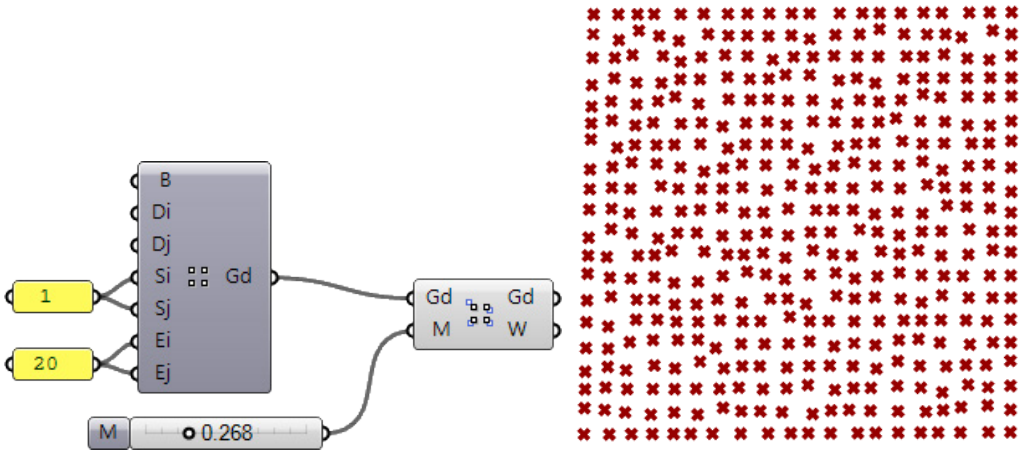
Gd: Resulting grid

W: Grid of weights (0-1)

Example

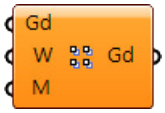
- Open ***..\PT_GH_PrimerExamples\Component Examples\ptRandAtts.gh***.

The example generates a rectangular grid and then shuffles the grid randomly. Notice that points are only shuffled within their local region, and the shuffling does not produce overlapped regions.



Weight Attraction (ptWeight)

The ***Weight Attraction (ptWeight)*** component feeds a weight field or grid and controls attraction directly. This is useful when you have defined the amount of attraction you want for each grid point.



Input

Gd: Input grid

W: Grid of weights (0-1)

M: Magnitude or degree of influence

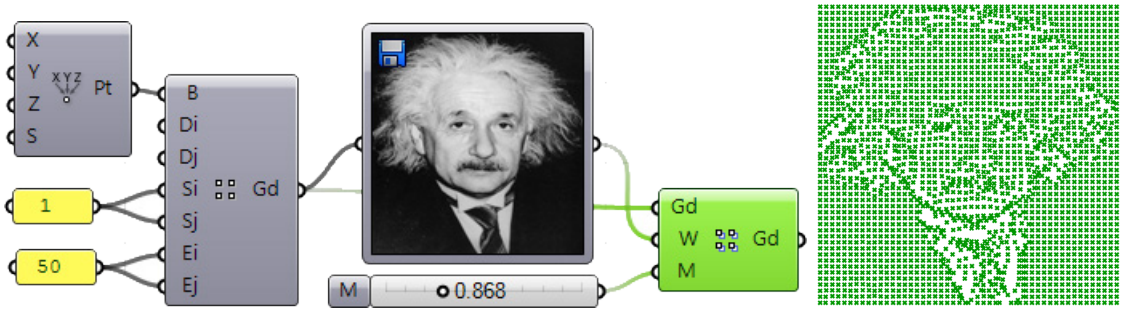
Output

Gd: Resulting grid

Example

- Open ***..\PT_GH_PrimerExamples\Component Examples\ptWeight.gh***.

The example uses a bitmap image to generate weights and then uses the image to shuffle points.



Chapter 6: Grid Utility Components

These components generate paneling grids and organize them in a tree structure where branches represent rows of points.

Center Grid (ptCenter)

The ***Center Grid (ptCenter)*** component extracts the center grid of another input grid.

Input

Gd: Input grid

S: Base surface [optional]

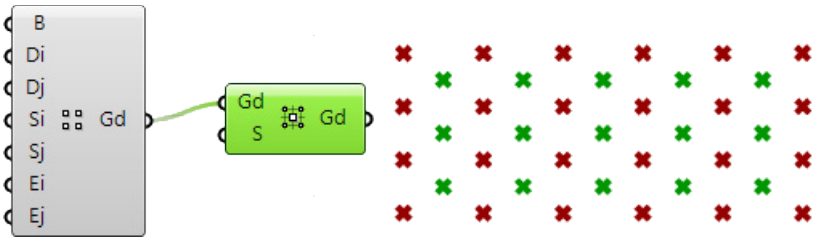
Output

Gd: Center grid

Example

Open ***..\PT_GH_PrimerExamples\Component Examples\ptCenter.gh***.

Use a rectangular grid as an input. The output is a grid of the centers of the cells.



Clean Grid (ptClean)

The ***Clean Grid (ptClean)*** component removes null rows and null columns in the input grid.

Input

Gd: input grid

Output

Gd: output grid

Convert to Diamond (ptToDiamond)

The ***Convert to Diamond (ptToDiamond)*** component converts rectangular grids to diamond grids.

Input

Gd: Input grid

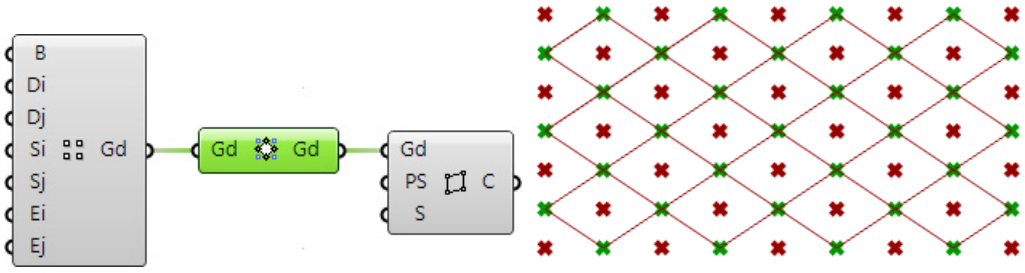
Output

Gd: Diamond grid

Example

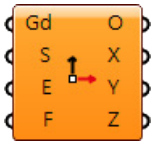
- Open ***..\PT_GH_PrimerExamples\Component Examples\ptToDiamond.gh***.

Use a rectangular grid as an input. The output is a diamond grid.



Coordinates (ptCoordinate)

The ***Coordinates (ptCoordinate)*** component calculates grid (or cell) coordinates, which are the origin, and x-, y-, and z-vectors for each grid/cell point. The x- and y-vectors are not normalized, and their length equals the distance between grid points.



Input

Gd: Input grid

S: Input surface [optional]

E: Calculate coordinates for end points [optional – set to *True* by default]

F: Flip the z-direction [optional – set to *False* by default]

Output

O: Origin points

X: x vectors

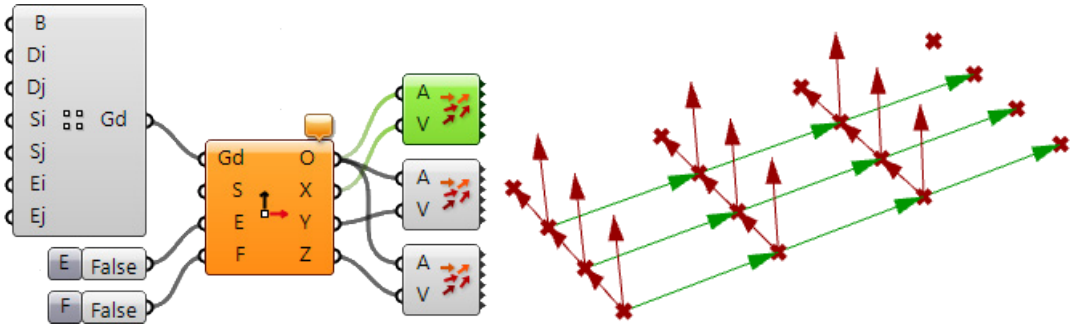
Y: y vectors

Z: z vectors

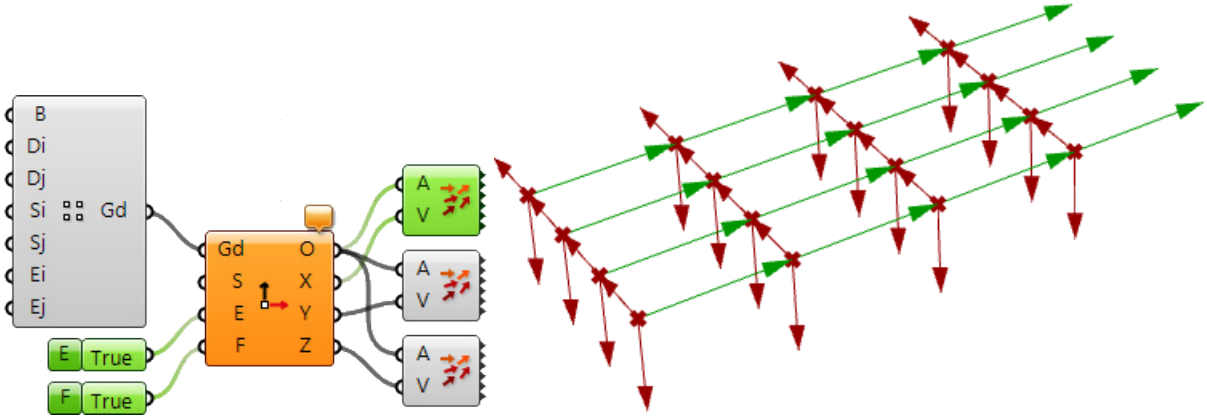
Example

- Open ***..\PT_GH_PrimerExamples\Component Examples\ptCoordinate.gh***.

The example shows the coordinates of each cell in the grid. If the surface *(S)* input parameter is not available, the component will calculate the surface using the input grid.

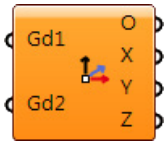


If the *E* input parameter is set to *True*, the coordinates for end points is calculated. The option to flip the z-direction is used in the example image.



Coordinates 3D (ptCoordinate3D)

The ***Coordinates 3D (ptCoordinate3D)*** calculates the box cell coordinates between two grids including the origin, and x-, y-, and z-vectors for each grid/cell point. The x, y, and z-vectors are not normalized and their length equals the distance between cell points.



Input

Gd1: First input grid

Gd2: Second input grid

Output

O: Origin points

X: x vectors

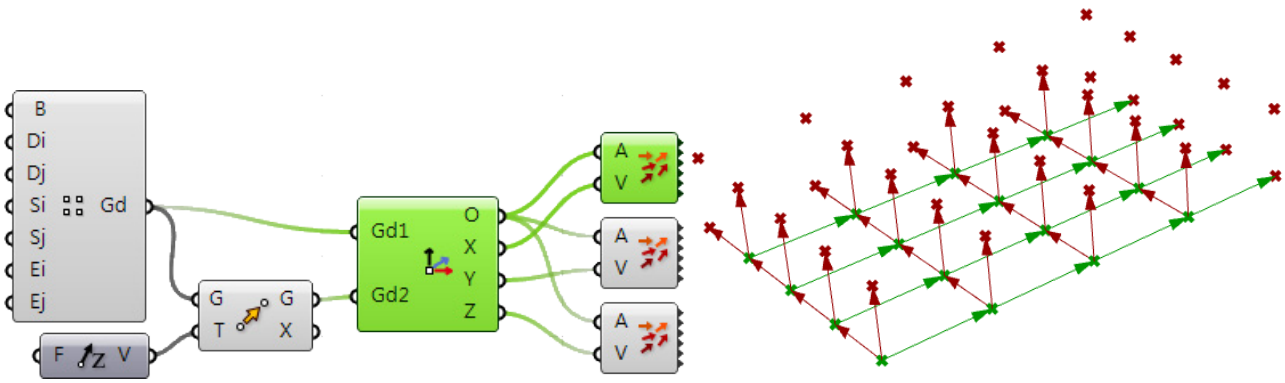
Y: y vectors

Z: z vectors

Example

- Open ***..\PT_GH_PrimerExamples\Component Examples\ptCoordinate3D.gh***.

The example creates a rectangular grid then copies it in the z-direction. The ***Coordinates 3D*** component takes the two grids as input and calculates x-, y-, and z-vectors for each cell.



Extract Column (ptCol)

The ***Extract Column (ptCol)*** component extracts columns from the grid.

Input

Gd1: Input grid

i: Column index

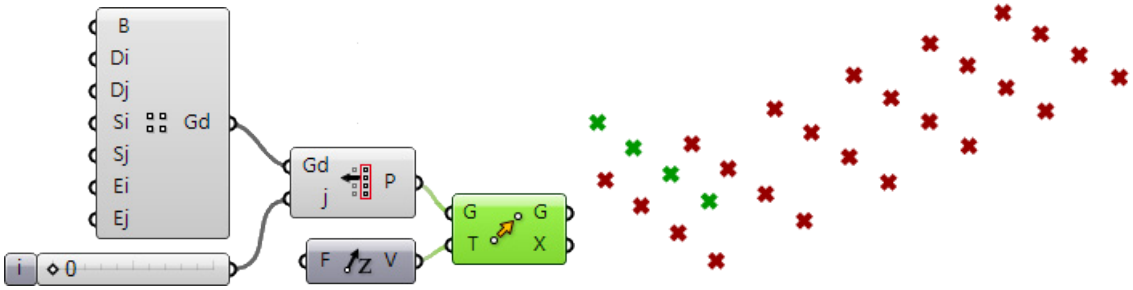
Output

P: List or grid of points

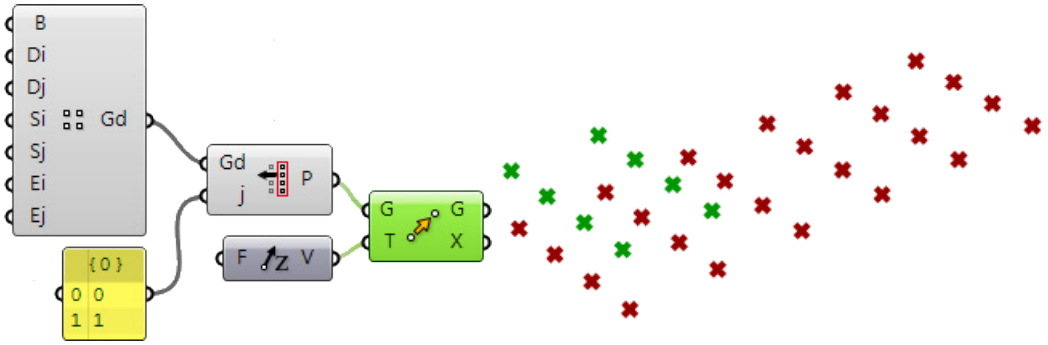
Example

Open ***..\PT_GH_PrimerExamples\Component Examples\ptCol.gh***.

The example extracts the first column of the grid and then moves it in the z-direction.

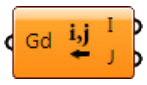


It is also possible to input a list of indices to extract multiple columns in the form of a grid as in the following:



Extract Grid Indices (ptIndices)

The ***Extract Grid Indices (ptIndices)*** takes an input a grid of points as input and outputs two grids of integers representing the *i* and *j* indices of the points. This component can be used to tag points.



Input

Gd: Input grid

Output

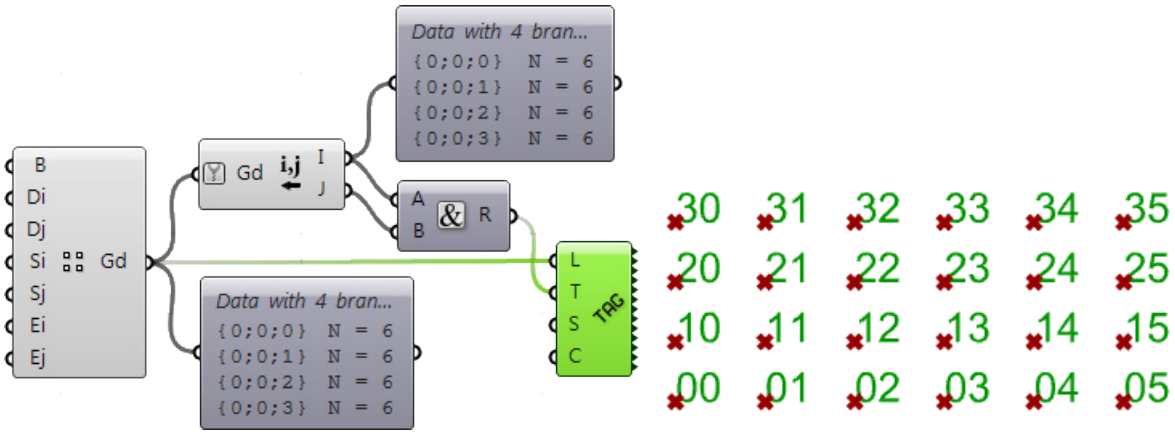
I: Grid of i indices

J: Grid of j indices

Example

Open ***..\PT_GH_PrimerExamples\Component Examples\ptIndices.gh***.

The example uses the ***Extract Grid Indices*** component to get the *i,j* indices of the points in a grid and then uses the Grasshopper ***Concatenate*** component to tag the points with their indices. The structure of the grid is the same as the indices.



Extract Item (ptItem)

The ***Extract Item (ptItem)*** component extracts a point (or list of points) in a grid given its *i* and *j* indices.

Input

Gd: Input grid

Output

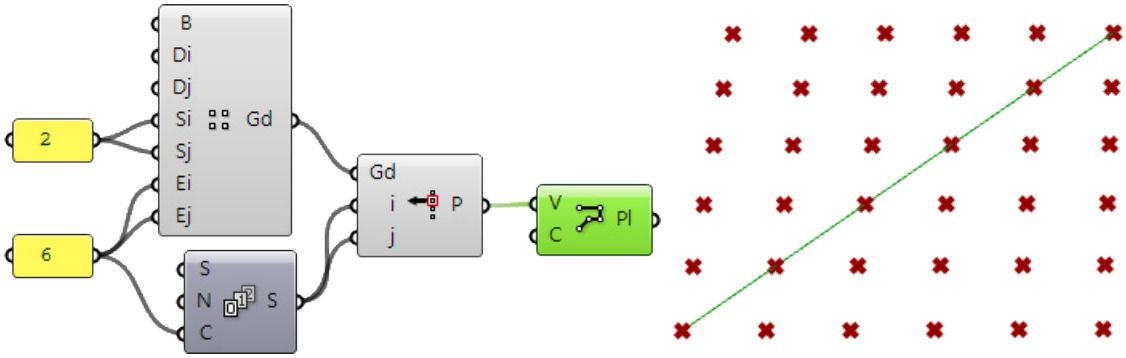
i: Item(s) i index

j: Item(s) j index

Example

Open ***..\PT_GH_PrimerExamples\Component Examples\ptItem.gh***.

The example uses the ***Extract Item*** component to extract diagonal points of a square grid then draws a polyline through them.



Extract Row (ptRow)

The ***Extract Row (ptRow)*** component extracts a row of points in a grid given the *i* index.

Input

Gd: Input grid

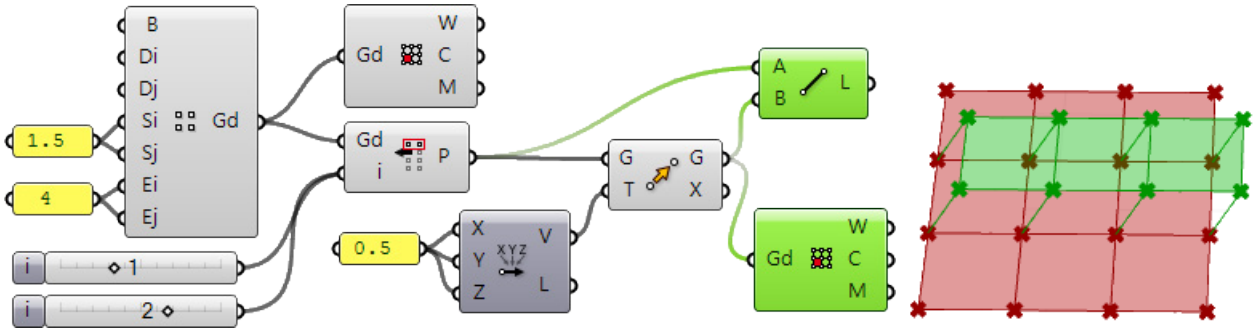
Output

i: Row index

Example

Open ***..\PT_GH_PrimerExamples\Component Examples\ptRow.gh***.

The example extracts the second and third rows of the grid and then moves these points diagonally, connects the points to the original rows with lines, and generates cells out of original and extracted grids.



Flatten Grids (ptFlatten)

The ***Flatten Grids (ptFlatten)*** component flattens a grid to a linear list of cells. This component reorganizes the grid so that the order of cells is linear and allows more control when mapping a list of modules to that grid.

Input

Gd: Input grid

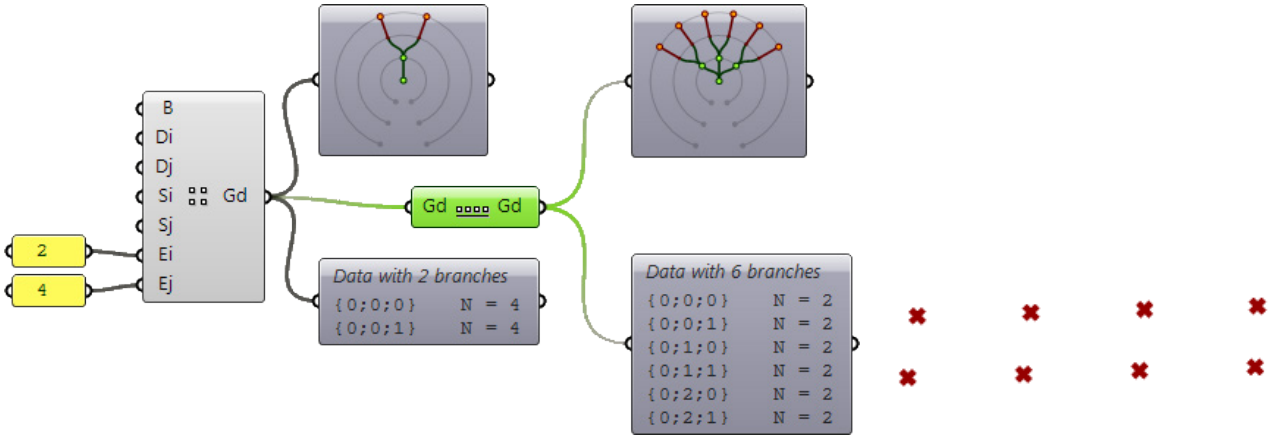
Output

Gd: Output grid of cells

Example

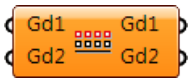
- Open ***..\PT_GH_PrimerExamples\Component Examples\ptFlatten.gh***.

The example shows how the grid is reformatted when put through the ***Flatten Grids*** component. The grid has two rows and four columns, so the initial tree has two branches with four elements in each branch. The flattened grid creates sub-trees equal to the number of cells in the grid with each sub-tree holding four elements organized in two rows and two columns.



Flatten 3D Grids (ptFlatten3D)

The ***Flatten 3D Grids (ptFlatten3D)*** component flattens two bounding grids into a linear list of bounding cells. The ***Flatten 3D Grids*** component reorganizes the grids so that the order of cells is linear and allows more control when mapping a list of modules to grid 3-D cells.



Input

Gd1: First input grid

Gd2: Second input grid

Output

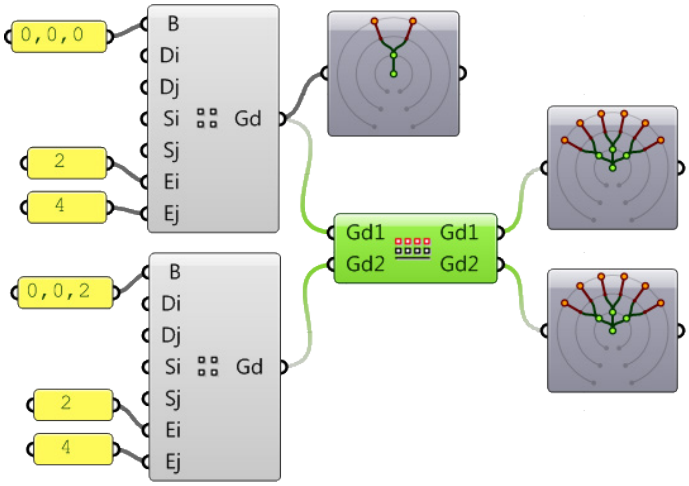
Gd1: Flattened first grid

Gd2: Flattened second grid

Example

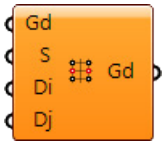
- Open ***..\PT_GH_PrimerExamples\Component Examples\ptFlatten3D.gh***.

The example shows how the grids are reformatted when put through the ***Flatten 3D Grids*** component. The input grids have two rows and four columns, so the initial tree has two branches with four elements in each branch. Flattened grids create sub-trees equal to the number of cells in each grid with the sub-trees holding four elements organized in two rows and two columns.



Grid Density (ptDense)

The ***Grid Density (ptDense)*** component either increases or decreases the density of the grid. An optional input surface makes sure added grid points lay on the surface.



Input

Gd: Input grid

S: Grid surface

Di: Change is row density. This can positive, negative or zero

Dj: Change is column density. This can positive, negative or zero

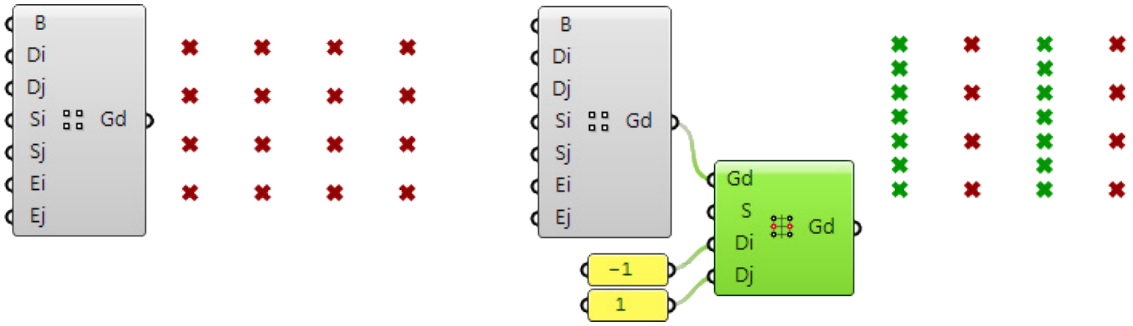
Output

Gd1: Output grid

Example

Open ***..\PT_GH_PrimerExamples\Component Examples\ptDense.gh***.

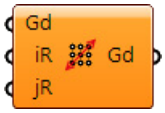
You can increase or decrease the density of a grid. In the example Di = -1. This removes every other element in each row. Setting Dj=1, inserts an additional grid point between each two column elements.



Before changing the density (left). After changing the density (right).

Grid Dir (ptDir)

The ***Grid Dir (ptDir)*** component reverses the grid row and column directions.



Input

Gd: Input grid

iR: Reverse row direction when set to *True*

jR: Reverse column direction when set to *True*

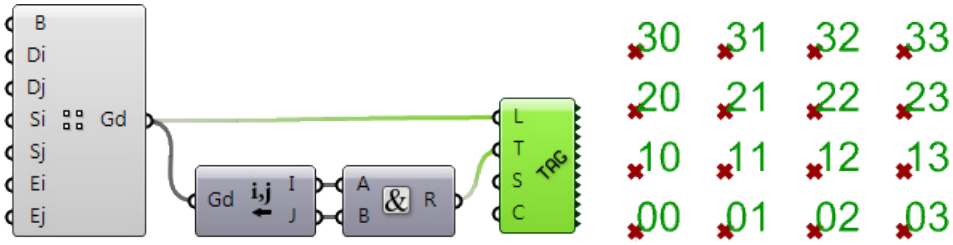
Output

Gd: Reversed grid

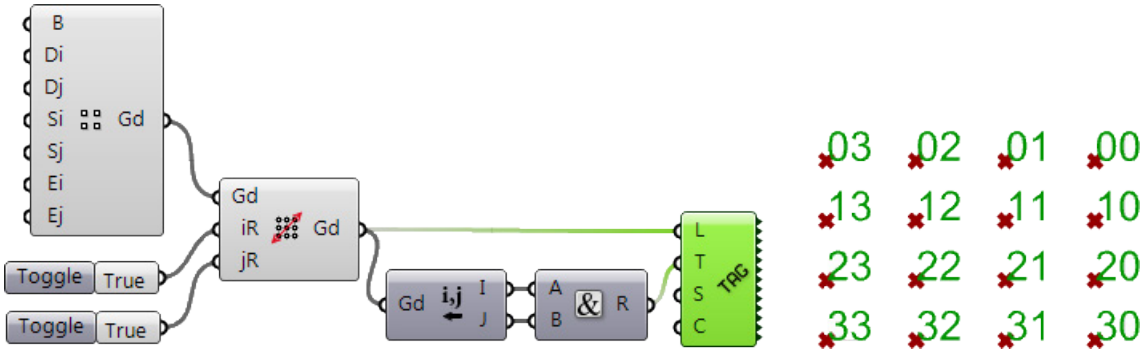
Example

Open ***..\PT_GH_PrimerExamples\Component Examples\ptDir.gh***.

The rectangular grid is tagged to show the row/column location of each grid point. Notice that the base point (00) is in the lower left point.

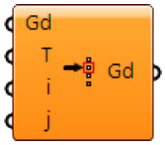


If we reverse the row and column directions, the base point (00) changes to the top right point.



Replace (ptReplace)

The ***Replace (ptReplace)*** component is used to replace one or more points in a grid.



Input

Gd: Input grid

T: List of one or more points

i: Corresponding row location of the point to be replaced

j: Corresponding column location of the point to be replaced

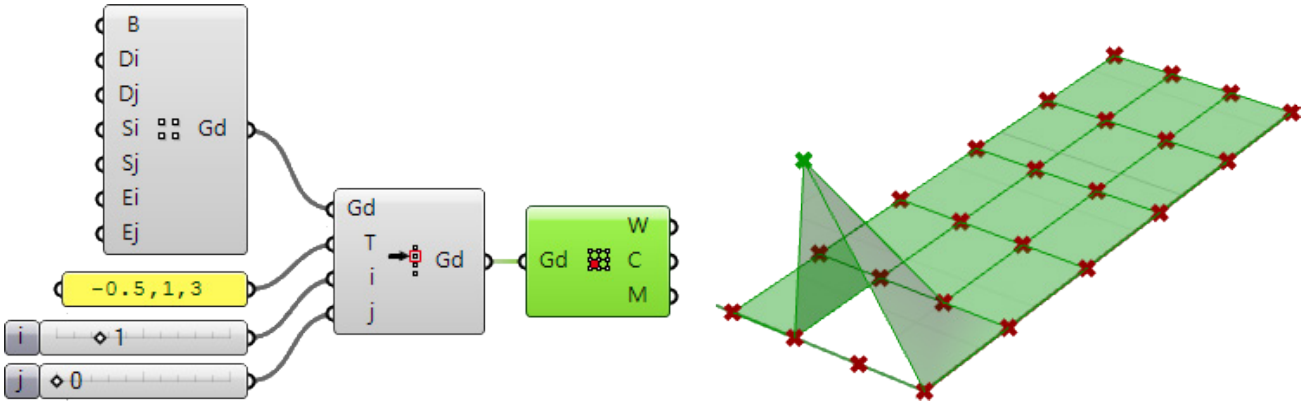
Output

Gd: Output grid

Example

- Open ***..\PT_GH_PrimerExamples\Component Examples\ptReplace.gh***.

The example shows how to replace one element in the grid. A ***Cellulate*** component was added to show the mesh of the new grid.



Square Grid (ptSquare)

The ***Square Grid (ptSquare)*** component turns a jagged grid of points into a rectangular grid filling the missing points with NULL points.

Input

Gd: Input grid

Output

Gd: Squared grid

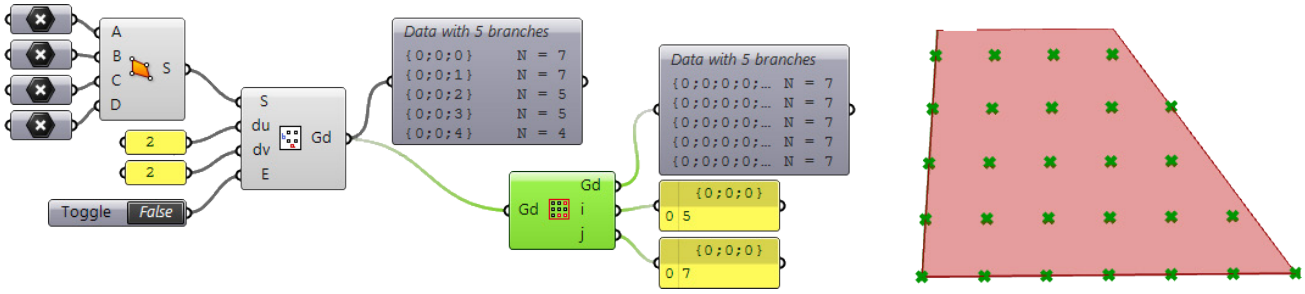
i: Number of rows

j: Number of columns

Example

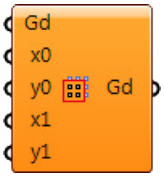
- Open ***..\PT_GH_PrimerExamples\Component Examples\ptSquare.gh***.

The example shows how to replace one element in the grid. A ***Cellulate*** component was added to show the mesh of the new grid.



SubGrid (ptSubGrid)

The ***SubGrid (ptSubGrid)*** component extracts part of the grid.



Input

Gd: Input grid

X0: Min x index for the sub grid

Y0: Min y

X1: Max x

Y1: Max y

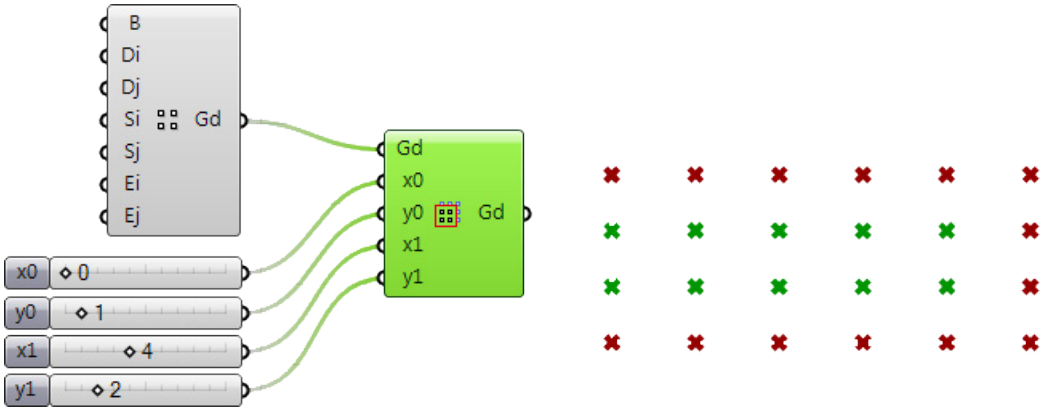
Output

Gd: Sub grid

Example

- Open ***..\PT_GH_PrimerExamples\Component Examples\ptSubGrid.gh***.

The example shows how extract a sub-grid dynamically using sliders.



Surface from Grid (ptGridSrf)

The ***Surface from Grid (ptGridSrf)*** component creates a NURBS surface from a given grid. The surface can be set to use grid points either as surface control points or to attempt to generate a surface that goes through the grid points. The former yields successful result more often.

Input

Gd: Input grid

T: When set to *True*, attempts to generate a surface that goes through the grid

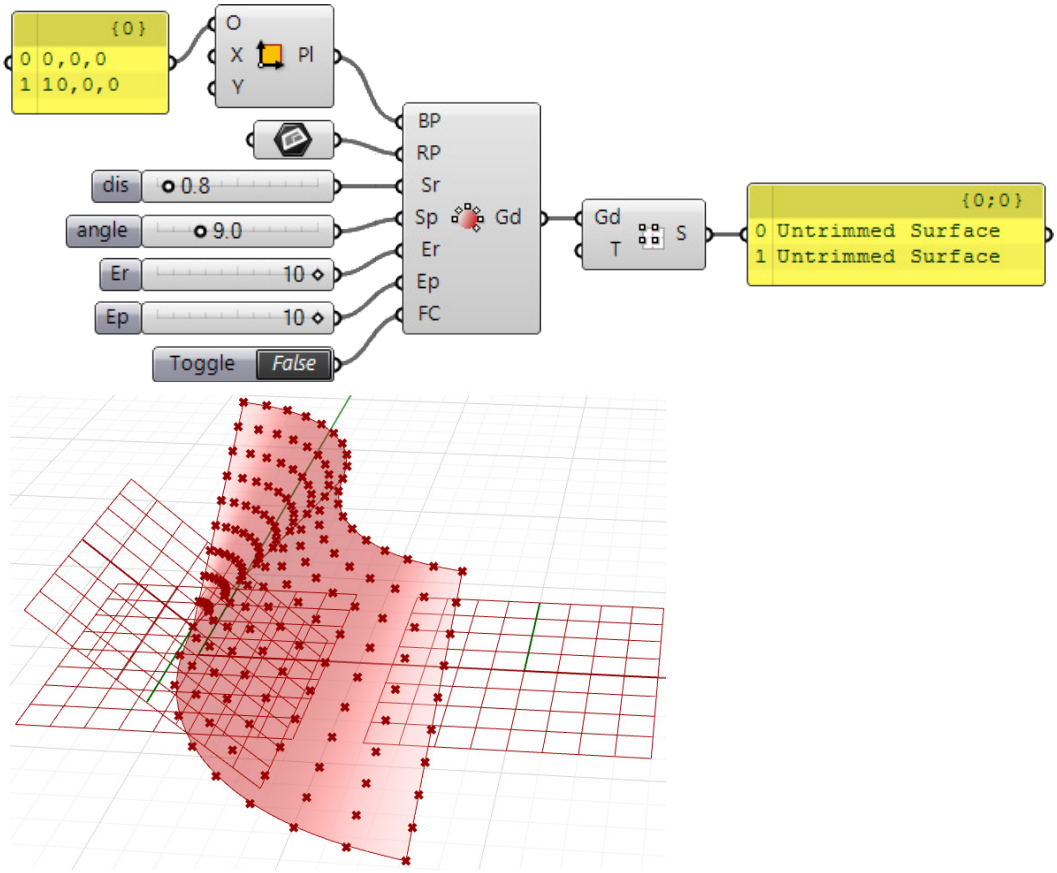
Output

S: Surface

Example

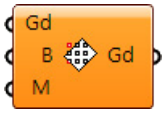
- Open ***..\PT_GH_PrimerExamples\Component Examples\ptGridSrf.gh***.

The example generates two surfaces using the two grids created by the ***Polar 3D Grid*** component



Trim Grid (ptTrim)

The ***Trim Grid (ptTrim)*** component trims a grid using base polysurface. The grid can be trimmed inside, outside, or points can be shifted out to the edge.



Input

Gd: Input grid

B: Trim polysurface

M: Trim method
0=inside
1=outside
2=edge

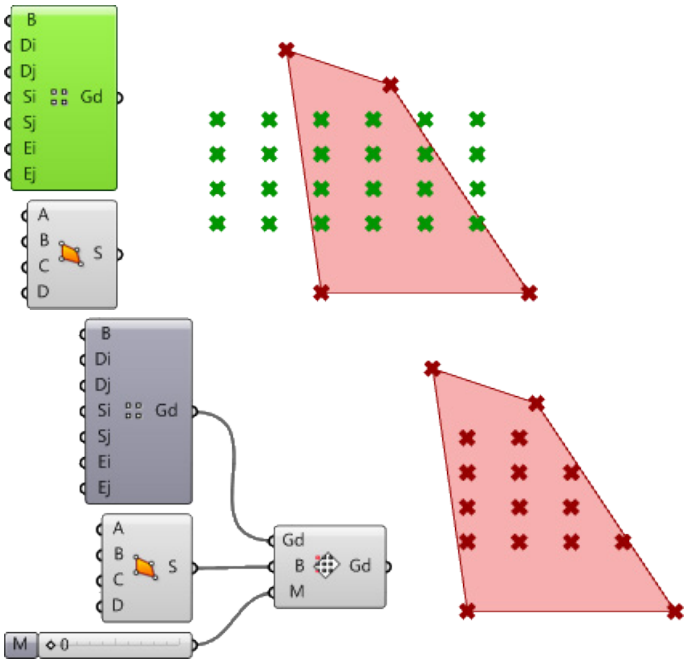
Output

Gd: Output grid

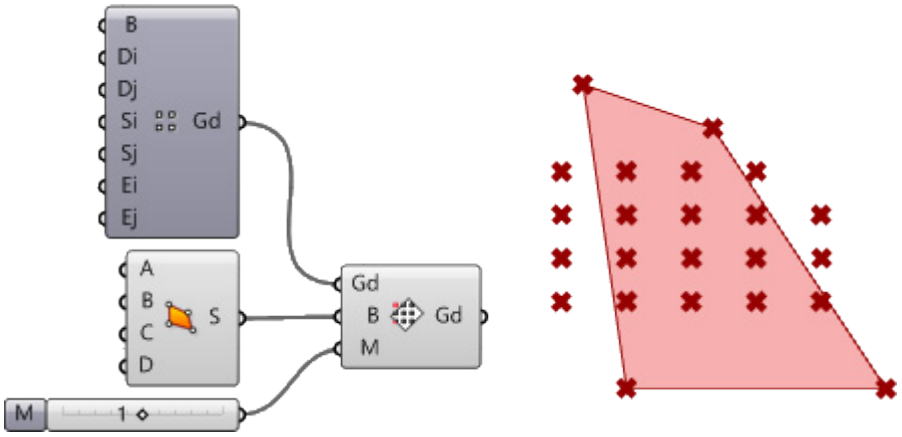
Example

- Open ***..\PT_GH_PrimerExamples\Component Examples\ptTrim.gh***.

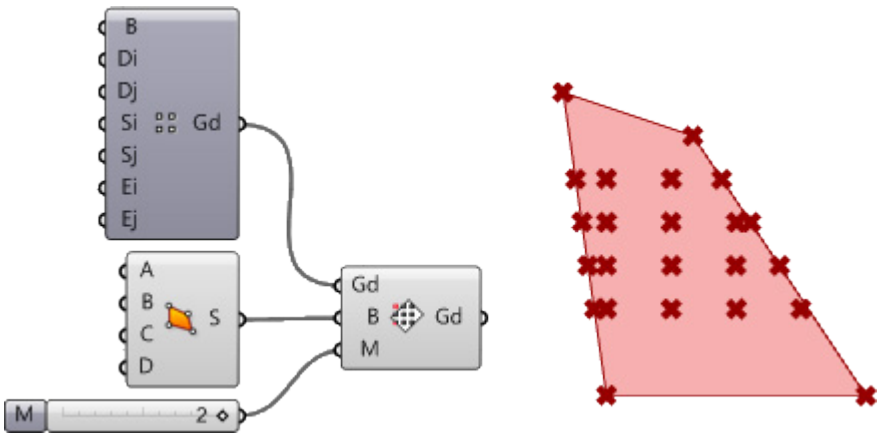
This example shows various types of trimming a reference surface.



Inside trimming. M=0.



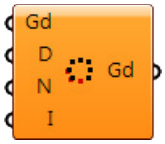
Outside trimming. M=1.



Edge trimming. M=2.

Wrap Grids (ptWrap)

The ***Wrap Grids (ptWrap)*** component copies rows or columns in place and appends to the end of the grid. This is sometimes needed to close a grid or extend far enough to accommodate patterns that stretch more than two grid points.



Input

Gd: Input grid

D: Direction
0=add rows
1=add columns

N: Number of columns/rows to wrap

I: Starting index of the row or column to copy in place

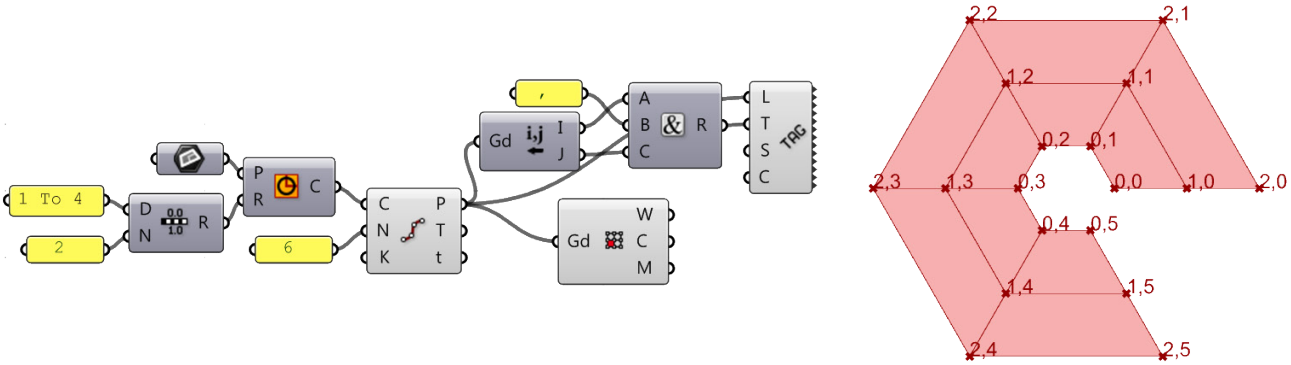
Output

Gd: Output grid

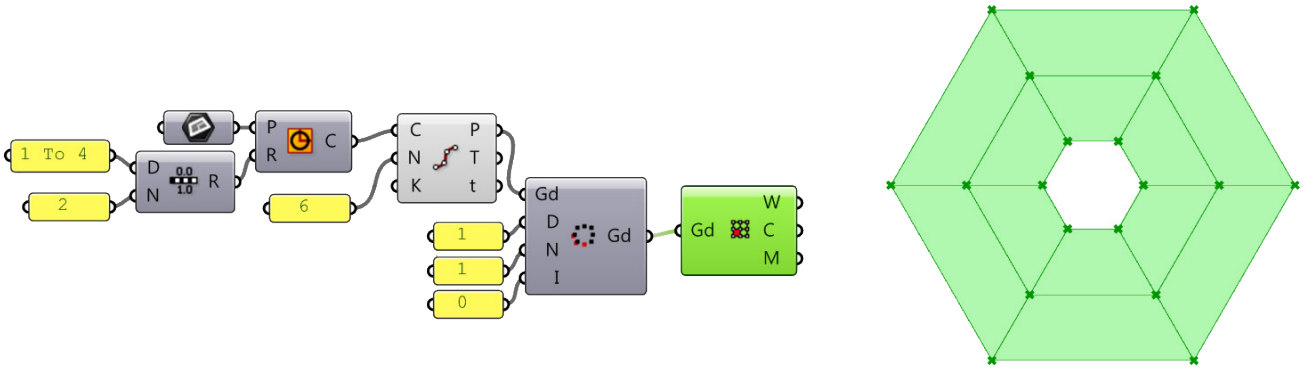
Example

- Open ***..\PT_GH_PrimerExamples\Component Examples\ptWrap.gh***.

The first definition shows a gap when adding panels. This is because the grid is not closed in the v- or y-direction, which means the first column of points needs to be appended to the end of the grid. For example, first row (labeled 0,0-0,1-…-0,5) needs an additional point (0,6) that overlaps (0,0) to close that row. The same goes for the remaining rows.



Adding the ***Wrap Grids*** component appends one more column at the end of the grid to close it.

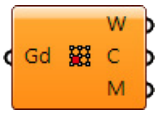


Chapter 7: Panel 2D Components

These components generate panels using one grid.

Cellulate (ptCell)

The ***Cellulate (ptCell)*** component generates list of wires, borders, and meshes of the paneling cells using a grid of points. It is possible to input multiple grids in which case, the output will be organized in main branches that represent the number of input grids.



Input

Gd: Input grid

Output

W: List of wires (4 wires per cell)

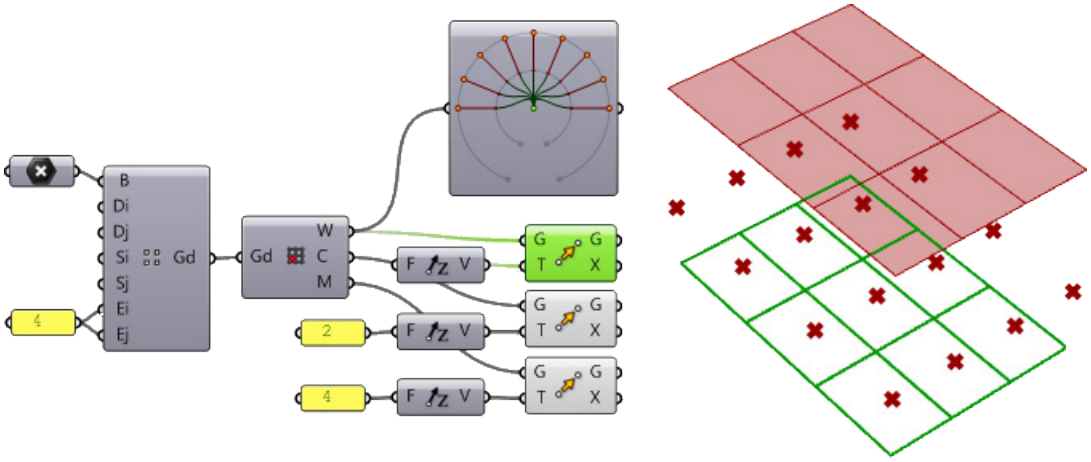
C: List of corners (4 points per cell)

M: List of meshes (one mesh per cell)

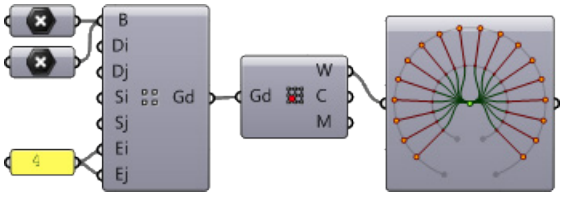
Example

- Open ***..\PT_GH_PrimerExamples\Component Examples\ptCell.gh***.

This example creates a 4x4 grid (9 cells). The ***Cellulate*** component generates lists of cell wires, cell corners, and meshes.

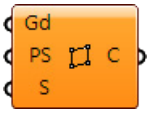


This example shows the output when the input is more than one grid.



Generate Borders (ptBorders)

The ***Generate Borders (ptBorders)*** component creates a structure of polycurve borders. The output is organized in branches that represent the rows of the paneling structure.



Input

Gd: Input grid

PS: Panel shape
0=straight
1=pull to the reference surface S
2=isocurve
3=shortest path

S: Reference surface. One is created from the grid if none is provided

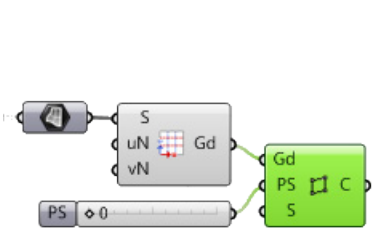
Output

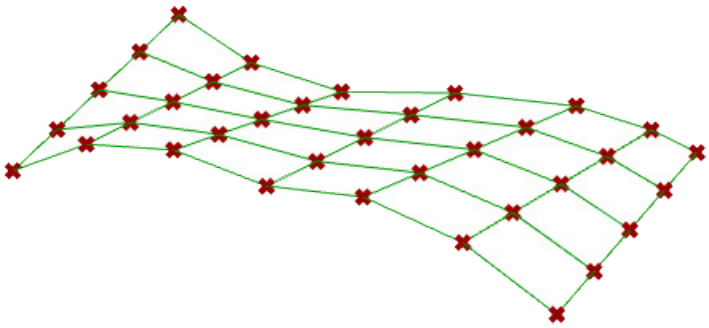
C: Output borders

Example

Open ***..\PT_GH_PrimerExamples\Component Examples\ptBorders.gh***.

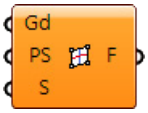
The output in this example has six branches with four curves (borders) in each branch.





Generate Faces (ptFaces)

The ***Generate Faces (ptFaces)*** component creates a structure of faces. The output is organized in branches that represent the rows of the paneling structure.



Input

Gd: Input grid

PS: Panel shape:
0=straight
1=pull to the reference surface S
2=iso
3=shortest path

S: Reference surface. One is created from the grid if none is provided

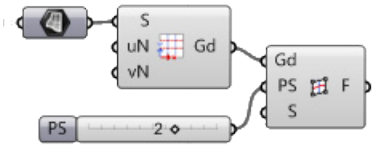
Output

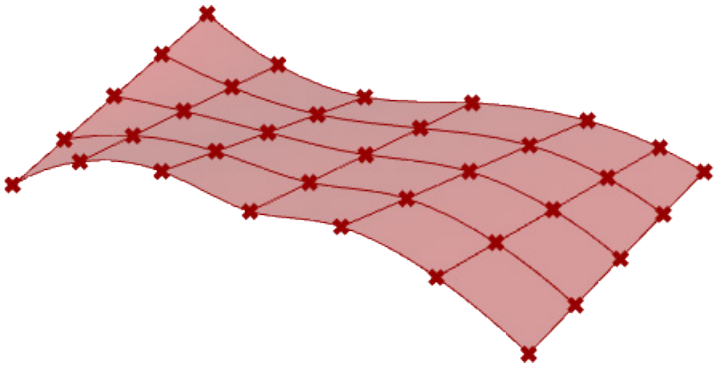
F: Output faces

Example

- Open ***..\PT_GH_PrimerExamples\Component Examples\ptFaces.gh***.

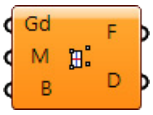
The output in this example has six branches with four faces in each branch.





Generate Flat Faces (ptFlatFaces)

The ***Generate Flat Faces (ptFlatFaces)*** component creates a structure of best-fit planar faces. The output is organized in branches that represent the rows of the paneling structure.



Input

Gd: Input grid

M: Flattening method
- 0=best fit planar faces
- 1=fit plane through (first, second, third) corners
- 2=fit through (second, third, fourth)
- 3=fit through (third, fourth, first)
- 4=fit through (fourth, first, second)

B: Optional reference base polysurface

Output

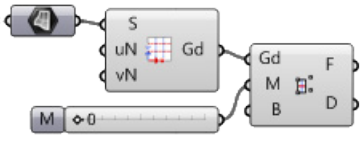
F: Output planar faces.

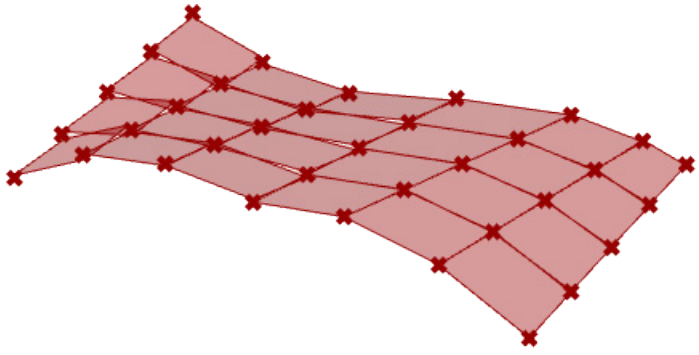
D: Deviation map of output faces from the grid

Example

- Open ***..\PT_GH_PrimerExamples\Component Examples\ptFlatFaces.gh***.

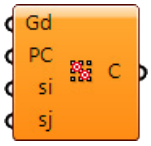
The output in this example has 6 branches with four planar faces in each branch. Faces might not join if the grid is free form, but they touch at least at one point and they do not overlap.





Morph 2D (ptMorph2D)

The ***Morph 2D (ptMorph2D)*** component distributes 2-D modules over a given paneling grid.



Input

Gd: Input grid

PC: Unit module (list of curves)

si: Shift in the i direction (between columns)

sj: Shift in the j direction (between rows)

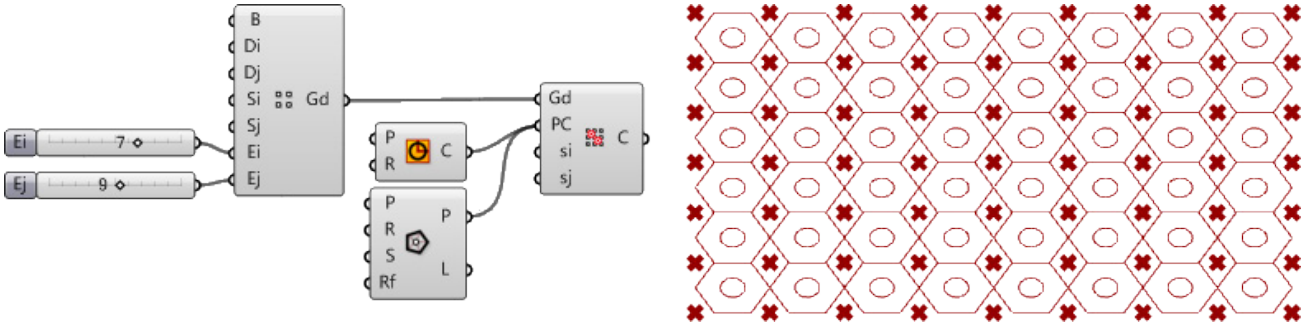
Output

Gd: List of output curves

Example

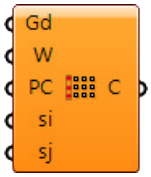
Open ***..\PT_GH_PrimerExamples\Component Examples\ptMorph2D.gh***.

The module in this example consists of two curves. Those are distributed on the grid using the ***Morph 2D*** component. The distributed unit module occupies the grid unit area bounded by four points.



Morph 2D List (ptMorph2Dlist)

The ***Morph 2D List (ptMorph2Dlist)*** component is a variable distribution of a list of modules using attractors.



Input

Gd: Input grid

W: Grid of normalized weights (values between 0 and 1) that has a structure identical to the input points grid

PC: List of pattern curves to be distributed following the weights map

si: Shift in the i direction (between columns)

sj: Shift in the j direction (between rows)

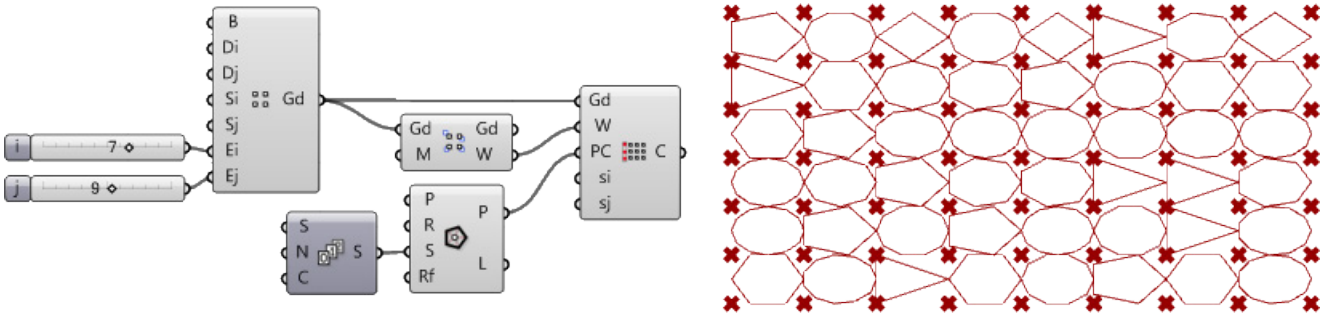
Output

C: List of output curves

Example

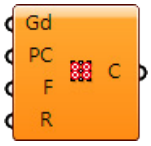
Open ***..\PT_GH_PrimerExamples\Component Examples\ptMorph2Dlist.gh***.

The input consists of ten polygons (with from 3 to 12 sides). Those are distributed to the grid randomly. The weights grid is generated using the ***Random Attraction*** component.



Morph 2D Map (ptMorph2DMap)

The ***Morph 2D Map (ptMorph2DMap)*** component maps each module in a list to one grid cell. The option to repeat the last module to the remaining cells is useful when specific modules should be applied to specific locations on the grid.



Input

Gd: Input grid

PC: List of pattern curves to be distributed in order

F: Flatten a grid into a list of cells

R: Repeat the last module. If the list of modules is less than the number of cells, use this option to map the last module to the rest of the grid cells

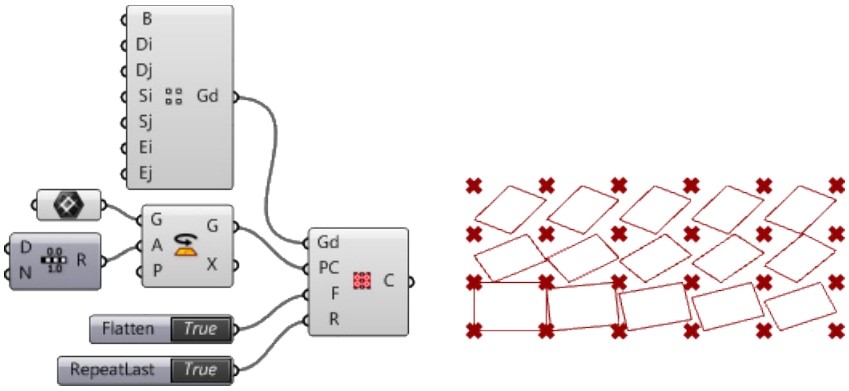
Output

C: List of mapped curves

Example

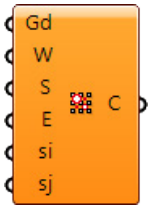
Open ***..\PT_GH_PrimerExamples\Component Examples\ptMorph2DMap.gh***.

In the example, we created 11 rectangles rotated slightly from one to the next. The input grid has 15 cells. Rectangles are mapped to the grid cells so as to repeat the last one to cover the rest of the cells.



Morph 2D Mean (ptMorph2DMean)

The ***Morph 2D Mean (ptMorph2DMean)*** component copies rows or columns in place and appends them to the end of the grid. This is sometimes needed to close a grid or extend the grid far enough to accommodate patterns that stretch more than two grid points.



Input

Gd: Input grid

D: Direction
0=add rows
1=add columns

N: Number of columns/rows to wrap

I: Starting index of the row or column to copy in place

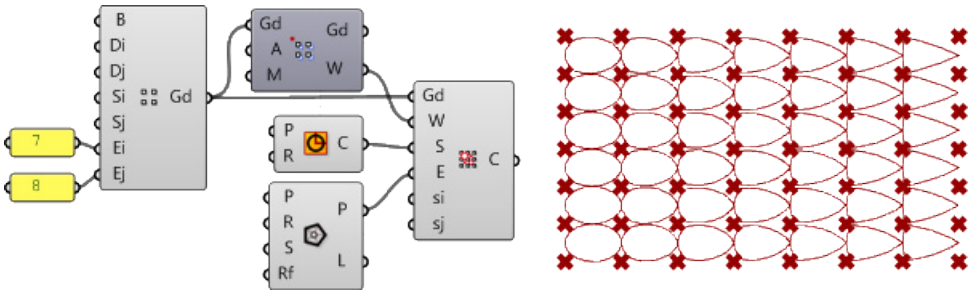
Output

Gd: Output grid

Example

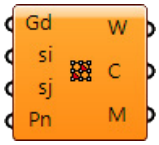
Open ***..\PT_GH_PrimerExamples\Component Examples\ptMorph2DMean.gh***.

Using the ***Point Attraction*** component, this example generates tween shapes between a circle and a triangle and maps them to the grid.



Panel Connections (ptMPanel)

The ***Panel Connections (ptMPanel)*** component generates fast and efficient pattern coverage through connecting grid points.



Input

Gd: Input grid

si: Shift in the i direction (between columns)

sj: Shift in the j direction (between rows)

Pn: Pattern string

Output

W: List of wires

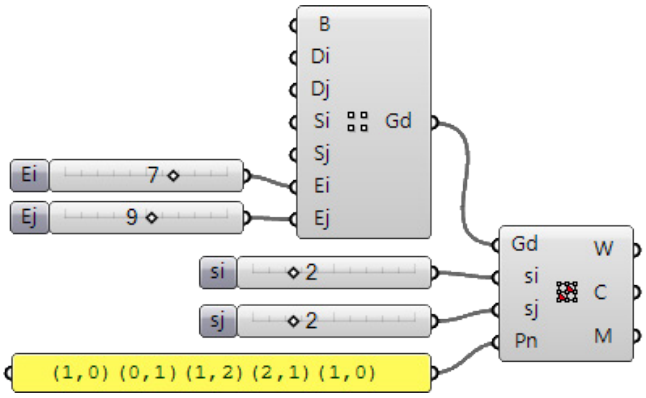
C: List of polycurves

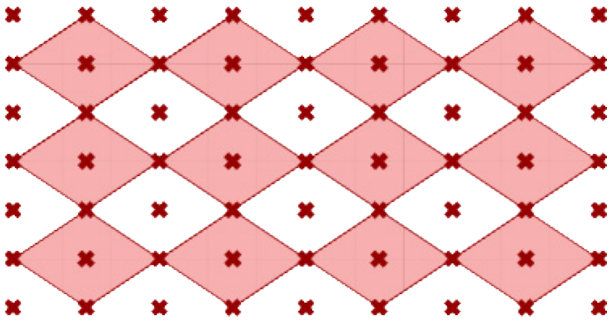
M: List of meshes

Example

Open ***..\PT_GH_PrimerExamples\Component Examples\ptMPanel.gh***.

This example creates a diagonal grid using ***Panel Connections*** component. The string defines unit connections assuming that the base point of the grid has (0,0) index. Shift in *i* and j is set to 2 because the diamonds span over 2 unit grids.



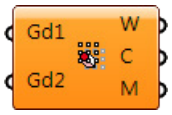


Chapter 8: Panel 3D Components

These components generate 3-D panels using two bounding grids.

Cellulate 3D Grid (pt3DCell)

The ***Cellulate 3D Grid (pt3DCell)*** component generates list of wires, cell corners, and meshes of the 3-D cell. The component takes two bounding grids and outputs list of wires connecting corresponding grid points, a list of corner points (8 points for each cell), and a list of mesh boxes.



Input

Gd1: First input grid

Gd2: Second input grid

Output

W: List of wires by rows

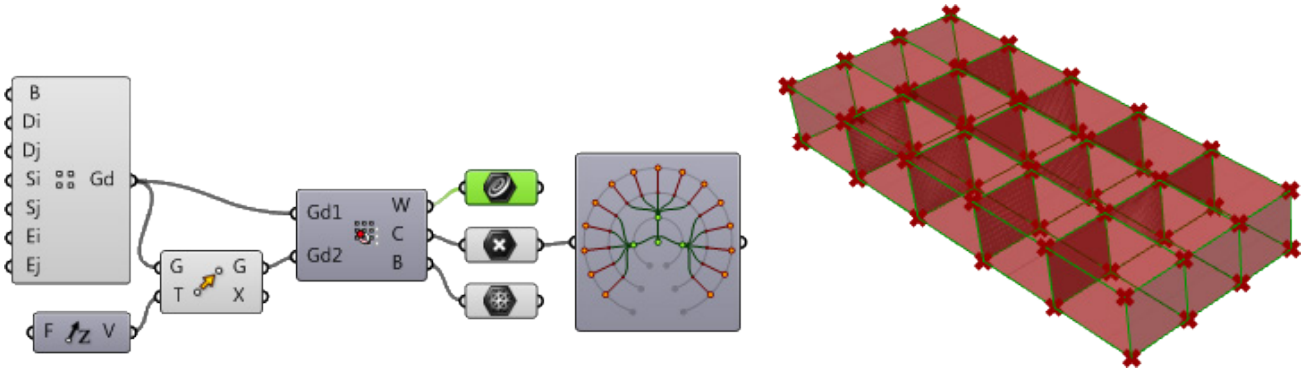
C: List of corners (8 points per cell)

M: List of mesh boxes by rows

Example

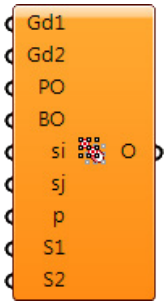
- Open ***..\PT_GH_PrimerExamples\Component Examples\pt3DCell.gh***.

The two bounding grids enclose 15 cells organized in three rows with five cells in each row. The output is organized by rows.



Morph 3D (ptMorph3D)

The ***Morph 3D (ptMorph3D)*** component morphs 3-D modules to 3-D cells enclosed by two bounding grids.



Input

Gd1: First input grid
Gd2: Second input grid
PO: Input pattern object
BO: Input bounding objects for pattern
si: Shift in unit grid in i-direction
sj: Shift in unit grid in j-direction
p: If set to *True* then perform smooth morphing
S1: First bounding surface (optional) corresponds to Gd1
S2: Second bounding surface (optional) corresponds to Gd2

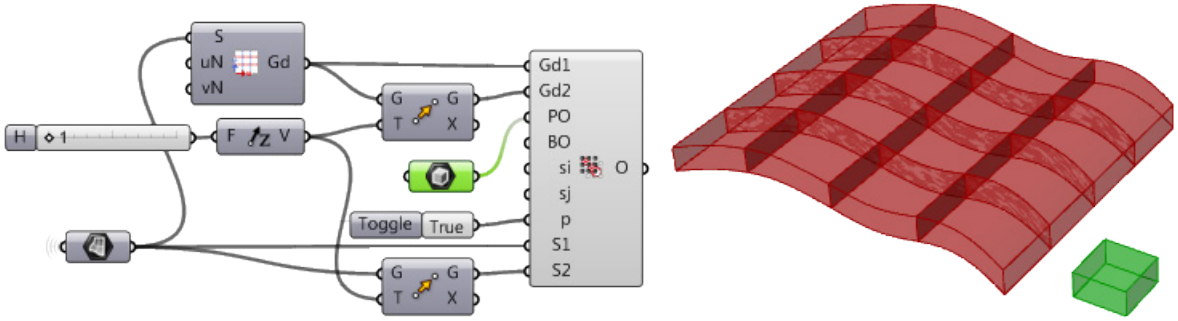
Output

O: Output morphed objects

Example

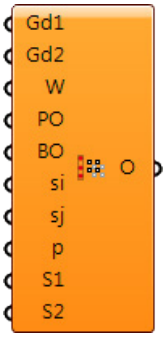
- Open ***..\PT_GH_PrimerExamples\Component Examples\ptMorph3D.gh***.

This example morphs a box smoothly between two bounding grids and surfaces.



Morph 3D List (ptMorph3Dlist)

The ***Morph 3D List (ptMorph3Dlist)*** component morphs 3-D list of modules to 3-D cells enclosed by two bounding grids.



Input

Gd1: First input grid
Gd2: Second input grid
W: Weight map
PO: Input list of pattern object
BO: Input bounding objects for pattern
si: Shift in unit grid in i-direction

sj: Shift in unit grid in j-direction
p: If set to *True*, perform smooth morphing
S1: First bounding surface (optional) (corresponds to Gd1)
S2: Second bounding surface (optional) (corresponds to Gd2)

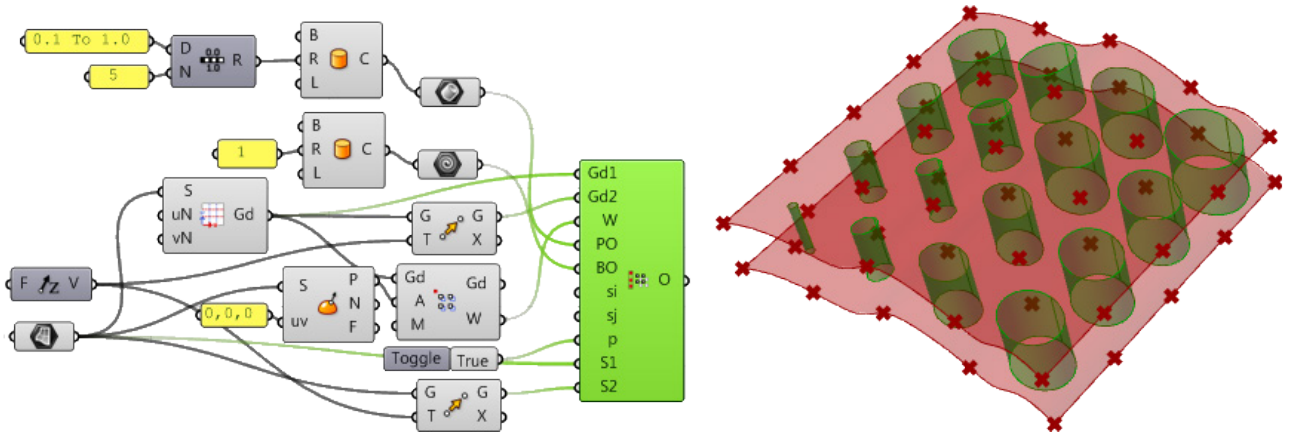
Output

O: Output morphed objects

Example

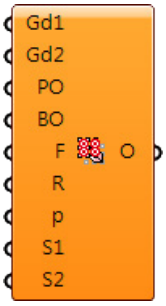
Open ***..\PT_GH_PrimerExamples\Component Examples\ptMorph3Dlist.gh***.

The example morphs a list of cylinders between two bounding grids and surfaces.



Morph 3D Map (ptMorph3DMap)

The ***Morph 3D Map (ptMorph3DMap)*** component maps each module in a list to one 3-D grid cell.



Input

Gd1: First input grid
Gd2: Second input grid
PO: Input list of pattern object
BO: Input bounding objects for pattern
F: Flatten the grid cells into one list of cells to be able to map in order
R: Repeat last object to remaining cells, if any
p: If set to *True* then perform smooth morphing
S1: First bounding surface (optional) (corresponds to Gd1)
S2: Second bounding surface (optional) (corresponds to Gd2)

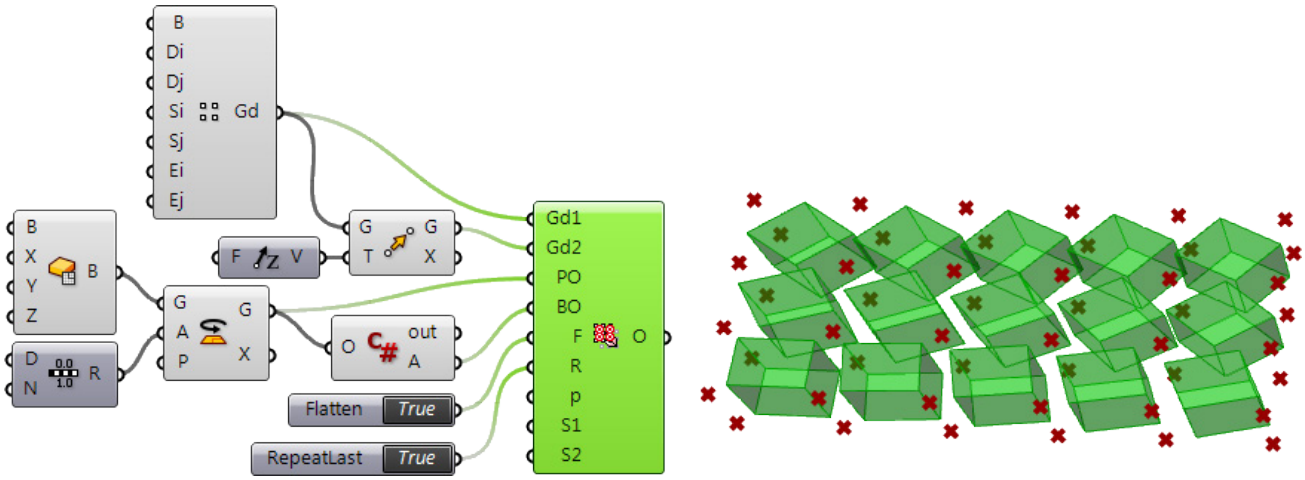
Output

O: Output morphed objects

Example

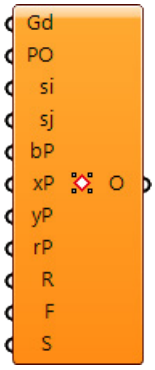
Open ***..\PT_GH_PrimerExamples\Component Examples\ptMorph3DMap.gh***.

The example morphs a list of rotated boxes between two bounding grids and surfaces.



Orient to Grid (ptOrient)

The ***Orient to Grid (ptOrient)*** component maps 2-D or 3-D modules to a grid. If pattern reference points are not provided, bounding box points of the pattern is used.



Input

Gd: Input grid

PO: Input pattern

si: Shift in unit grid in i-direction

si: Shift in unit grid in i-direction

bP: Base point for the module

xP: X direction reference point

yP: Y direction reference point

rP: Fourth reference point

R: Rigid orient

F: Flip

S: Base surface for smooth morph (optional)

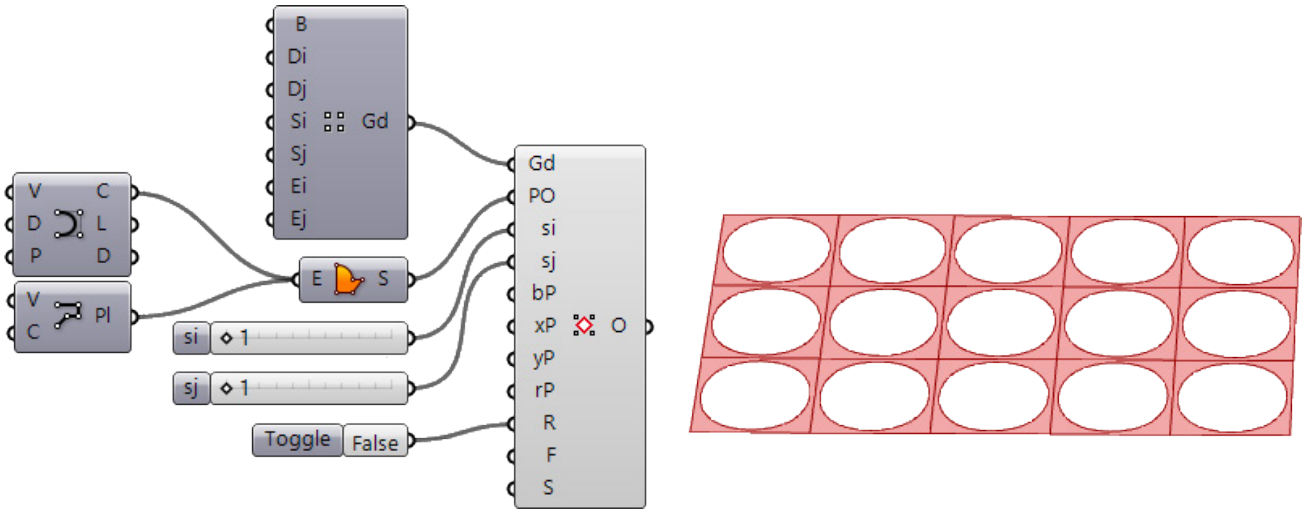
Output

O: Output morphed objects

Example

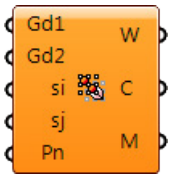
Open ***..\PT_GH_PrimerExamples\Component Examples\ptOrient.gh***.

The example morphs a trimmed surface to a grid.



Panel 3D Grid (ptMPanel3D)

The ***Panel a 3D Grid (ptMPanel3D)*** component creates 3-D paneling using modules defined by connecting grid points.



Input

Gd1: First input grid

Gd2: Second input grid

si: Shift in unit grid in i-direction

sj: Shift in unit grid in j-direction

Pn: Pattern string

Output

W: Line wires

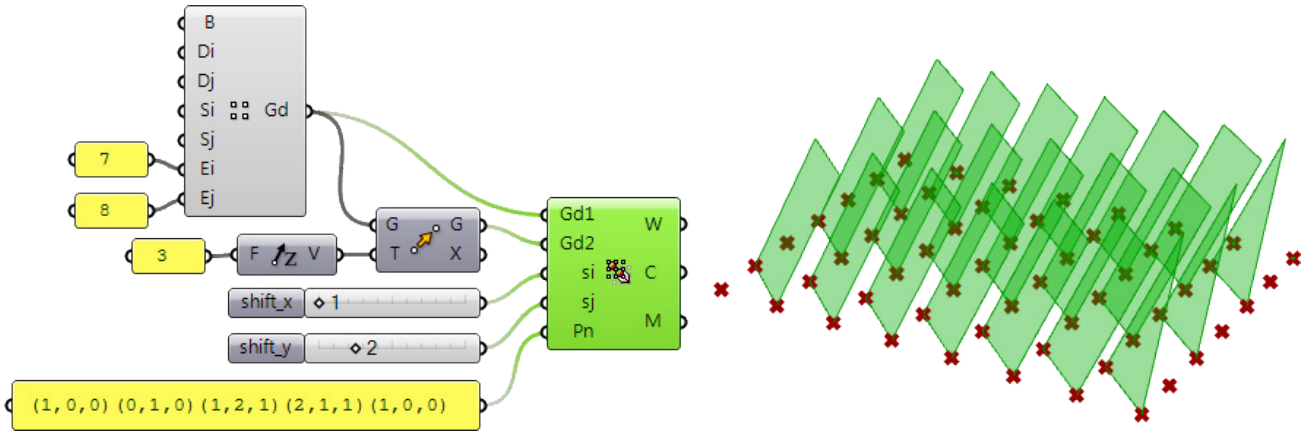
C: Polylines

M: Meshes

Example

- Open ***..\PT_GH_PrimerExamples\Component Examples\ptMPanel3D.gh***.

This example creates a custom module and populates it between two bounding grids.

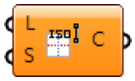


Chapter 9: Panel Utility Components

These components help create special panels.

Iso Edges (ptIsoE)

The ***Iso Edges (ptIsoE)*** component helps extract iso-edges on a given surface using linear edges. It uses the two end points of the input lines and generates an isocurve on the surface using the surface uv-directions. Both end points need to lie on the surface and align on the same surface isocurve.



Input

L: Lines to extract the isocurves

S: Input surface

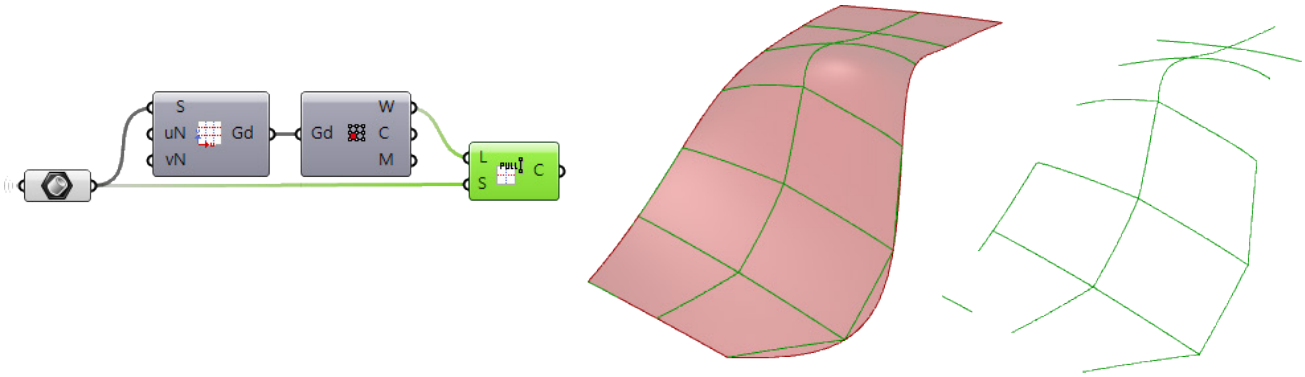
Output

C: Resulting isocurves

Example

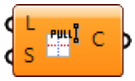
- Open ***..\PT_GH_PrimerExamples\Component Examples\ptIsoE.gh***.

If pulled straight edges do not end up on surface, edge curves can be missing or short.



Pull Edges (ptPullE)

The ***Pull Edges (ptPullE)*** component pulls linear edges to a surface.



Input

L: Lines to extract the isocurves

S: Input surface

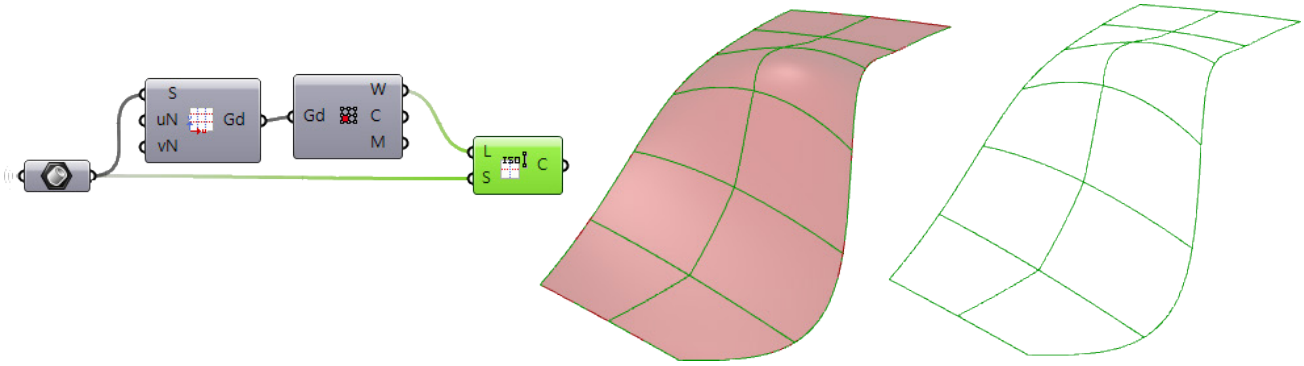
Output

C: Resulting pulled curves

Example

Open ***..\PT_GH_PrimerExamples\Component Examples\ptPullE.gh***.

When the two end points of the line segments align with the surface u and v, the result is clean continuous curves.



Short Edges (ptShortE)

The ***Short Edges (ptShortE)*** component extracts shortest path on surface between the two end points of input linear edges.

Input

L: Lines to extract the iso curves

S: Input surface

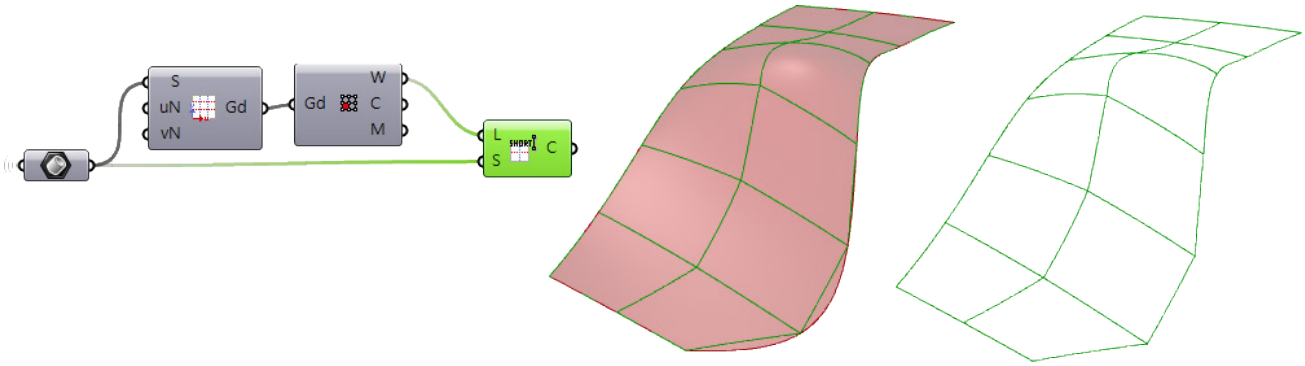
Output

C: Resulting short curves

Example

Open ***..\PT_GH_PrimerExamples\Component Examples\ptShortE.gh***.

If the two end points are within the surface domain, the shortest path will always result in a complete curve, but will not necessarily follow the iso-direction of the surface.



Chapter 10: Select and Bake Grids Components

PanelingTools is an integrated plugin for Rhino and Grasshopper. Sometimes it is useful to go back and forth between the Rhino and Grasshopper environments.

Bake Grid (ptBakeGrid)

The ***Bake Grid (ptBakeGrid)*** component saves grids from Grasshopper as Rhino geometry.

Input

G: Input grid(s)

S: Name of the grid (string)

B: Bake the grid when set to *True*. **Note**: Set the **Toggle** component back to *False* once the geometry is saved or a new set of geometry will be created each time the component is used.

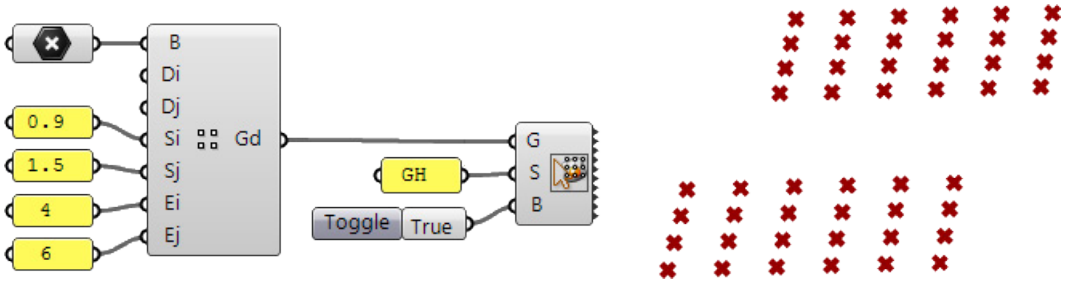
Output

Paneling grid(s) saved to Rhino as geometry.

Example

- Open ***..\PT_GH_PrimerExamples\Component Examples\ptBakeGrid.gh***.

You can bake more than one grid from Grasshopper to Rhino.



Select Grid (ptSelGrid)

The ***Select Grid (ptSelGrid)*** component brings grids created and edited in Rhino into Grasshopper.

Input

Click on the icon in the component to select a paneling grid created with PanelingTools.

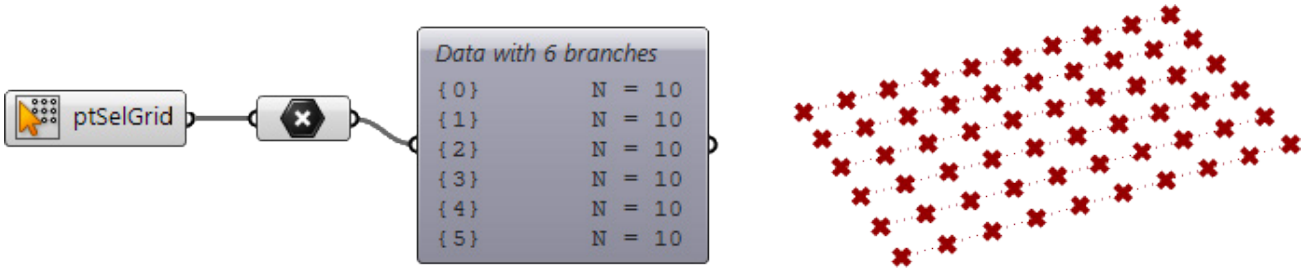
Output

Ordered grid in Grasshopper

Example

- Open ***..\PT_GH_PrimerExamples\Component Examples\ptSelGrid.gh***.

Paneling grids created in Rhino sustain their structure when selected into Grasshopper as shown in the following.



PT Version (ptVersion)

The ***Version (ptVersion)*** component displays the current version of PanelingTools for Grasshopper.